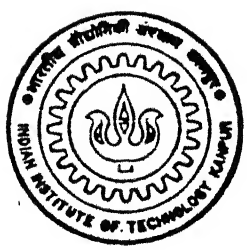
SOME STUDIES ON BETA - SKELETONS

by

S. V. Rao



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Indian Institute of Technology, Kanpur

JULY, 1998

SOME STUDIES ON BETA-SKELETONS

A Thesis Submitted

in Partial Fulfilment of the Requirements

for the Degree of

Doctor of Philosophy

by

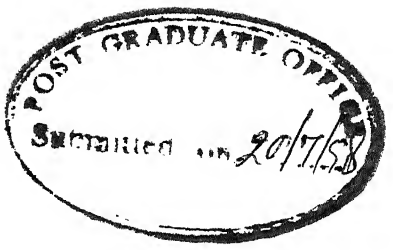
S. V. Rao

to the

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

July, 1998



CERTIFICATE

Certified that the work contained in the thesis entitled "*SOME STUDIES ON BETA-SKELETONS*", by "*S. V. Rao*", has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

(Prof. Asish Mukhopadhyay)
Department of Computer Science & Engg.,
Indian Institute of Technology,
Kanpur.

July, 1998

Synopsis

The present thesis is a study of some algorithmic aspects of β-skeletons. The concept of a β-skeleton was introduced by Kirkpatrick and Radke [KR85] as a way of capturing the "internal shape" of a planar set of points. Indeed, we get an entire spectrum of shapes by varying the parameter β. For a fixed value of β, the "internal shape" is a geometric graph, obtained by joining each pair of points whose β-neighborhood is empty.

For $\beta \geq 1$, the β-neighborhood of a pair of points p_i, p_j is the **interior** of the intersection of two circles of radius $\beta|\overline{p_ip_j}|/2$, centered at the points $(1-\beta/2)p_i + (\beta/2)p_j$ and $(\beta/2)p_i + (1-\beta/2)p_j$, respectively. For $\beta \in [0,1]$, it is the **interior** of the intersection of two circles of radius $|\overline{p_ip_j}|/(2\beta)$, passing through p_i and p_j. We denote this neighborhood by $lune_\beta(p_i, p_j)$. The parameter β characterizes the neighborliness of a pair of points.

Let $P = \{p_1, p_2, \ldots, p_n\}$ be a set of n points in the plane. The **β-skeleton** of P is a geometric graph (P, E) such that for every $p_i, p_j \in P$, the edge $\overline{p_i, p_j} \in E$ if and only if the β-lune $lune_\beta(p_i, p_j)$ is empty.

We discuss a sweepline algorithm for constructing a β-skeleton for any $\beta \geq 0$. The time-complexity of our algorithm is in $O(n^{3/2} \log n)$ for $\beta \geq 1$ and in $O(n^{5/2} \log n)$, for $\beta \in [0,1)$ under the metric L_p for $1 < p < \infty$. In the metrics L_1 and L_∞, our algorithm takes $O(n^{5/2} \log n)$ time for any $\beta \geq 0$.

The concept of a β-skeleton have natural generalizations to that of $k\beta$-skeletons and additively weighted β-skeletons. We have shown that the above sweepline method can be extended to construct these geometric graphs too. The time complexity of computing the $k\beta$-skeleton is in $O(kn^{3/2}\log n + k^2 n)$ for $\beta \geq 1$ and in $O(n^{5/2}\log n + n^2 k)$ for β in the range $[0, 1)$. The time complexity of computing the weighted β-skeleton is in $O(n^{3/2}\log n)$ for $\beta \geq 1$ and in $O(n^{5/2}\log n)$ for β in the range $[0, 1)$.

We have obtained an optimal output-sensitive algorithm for computing a β-skeleton for any $\beta \geq 2$ in metric L_1 and L_∞. The time complexity is in $O(n \log n + k)$, where k is the size of the β-skeleton.

We have presented efficient algorithms for computing the entire spectrum of β-shapes in L_2 metric. This means computing for each pair of points the largest β value, β_{max}, for which the β-neighborhood is empty. Our algorithms are in $O(n^{3/2}\log\ n + K)$ for β_{max} values not less than 1 and in $O(n^2 \log\ n + K)$ time for values greater than or equal to 0. A point that lies in the β-neighborhood of an edge is a witness to this edge. The quantity K is a count of the number of times that the edges of the initial graph on P are witnessed for $\beta = \infty$. We have also presented algorithms for computing the spectra of of $k\beta$-shapes and additively-weighted β-shapes.

We have extend the definition of β-skeleton into higher-dimensions. We have shown that the size of the β-skeleton for $\beta > 2$ is linear in d-dimensional space, for any fixed d. We have presented an efficient algorithm for computing the β-skeletons for $\beta > 2$.

Acknowledgments

It is pleasure to thank my thesis supervisor Prof. Asish Mukhopadhyay for constant support and encouragement given to me. It has been a privilege to work with him. Learning how to do research and how to write technical reports from him has being a life time experiences in itself. I thank my programme advisor, Dr. P. Gupta, for his kind cooperation in final stages of my programme.

I would like to thank Hari, Kali, TSR, Keswa, SB Rao, N. D. Reddy, Rajan, Jitendra Das, PVM Rao, PVK Reddy, MKV, Subram, Parasar, Parabal, Pranab, Kamaljith, Brhamaji, Amit, Chitali, Malviyka, Patnayak, Murali, Amit Das, Devanand, Kusum, Samudra, and Didi, who made my stay more enjoyable. I thank colleagues Sajith, Gore, Suresh, Veena, Atul, and Kshitiz for their company. I am thankful to all the faculty and staff for their cooperation and providing various facilities.

I thank my parents for leaving me alone to persue higher studies. Finally, I thank my wife, Sailaja, for her kind cooperation.

S. V. Rao

Contents

List of Figures

Chapter 1

Introduction

Shape is only operationally defined. Thus things do not have a shape the way Santa Claus has a red suit - Jan Koenderink in Solid Shape [Koe90]

In many applications in image processing, computer vision, pattern recognition, computational morphology, and geometric modeling *et al*, an object is specified in terms of a set of points. The shape reconstruction and subsequent identification of this object from the set of input points turns out to be one of the main problems confronting researchers working in these areas. Similarly, in spatial analysis [GB78, HCF77], the locational identifiers of some real spatial phenomena are represented by a set of points. Given the input points, the problem here is to understand and characterize the real phenomena.

A set of points, however, is an ambiguous representation of an object or a real phenomenon and, therefore, cannot be used directly in many applications. On the other hand, though the whole boundary can be used to define an object unambiguously, such a description would take up a huge amount of storage and a lot of processing time. To optimize the use of resources, we therefore need efficient methods to extract the object shape from the point set and fast methods to perform various operations on the reconstructed object.

The complexity of reconstructing the shape of an object from a set of points also depends on the point extraction method. For example, the problem becomes harder if the input points are unorganized. In many applications, the point extraction method is not known and input points are unorganized. So an unified method for reconstructing the object shape is required to handle data from various sources.

In some applications, the input primitives are more complex then points, through not unorganized. For example, a well-known problem in the field of Medical Imaging is to reconstruct a surface from a set of parallel cross-sectional data [BS94, GKDM96, AJM97].

It is known that, in many applications geometric properties alone cannot determine the shape of an object [Tou80b]. Even the exact solution of the geometric problems gives only partial, approximate, or heuristic solutions to various application problems. Shape reconstruction using non-geometric features of points is beyond the scope of the thesis.

The problem of shape recovery that we have discussed so far gives rise to the closely-associated question of what if anything is meant by the shape of a set of points. The question is important because the tools available for the shape recovery problem that we have discussed so far depends crucially on our answer to this question.

There is no single definitive answer. One reason for this is that there is no one definition that is best for all applications. So various definitions are extant in the literature. For example, Kirkpatrick and Radke [KR85] have defined the shape of a set of points as the decomposition of a point set into components that identify "essential" points and, among these, join "essential" neighbors. As is apparent, this definition depends on the interpretation of the term "essential".

Shape descriptors are divided into two categories: those that capture, so to say, the "external" shape of a set of points as opposed to those that capture what is

called the "internal" shape [KR85]. We explore these definitions in more details in the following sections.

1.1 External Shapes

> *The external shape of a set of points is obtained by identifying the "essential" extreme points of the set and, among these, joining "essential" neighbors* [KR85].

Among the simplest structures that capture the external shape of a planar set of points are the **minimum bounding box** and the **minimum spanning circle** of this set. The minimum bounding box is the smallest rectangle which is parallel to the axies and encloses all the points. This can be computed in $O(n)$ time. The minimum spanning circle is the smallest circle that encloses all the points. This can be computed in linear time [Meg83]. Instead of a **minimum spanning circle** one could consider as an alternative the **minimum spanning ellipse**. This can be computed in $O(n^2)$ time [Pos81, Pos84].

In the direction of increasing descriptive complexity and refinement, next to the above shape descriptors is the **convex hull** of the given point set. The convex hull is the smallest convex set that encloses all the points. Computing the convex hull of a point set is one of the most well-researched problems in computational geometry [PS85]. Various properties of the convex hull are used in pattern recognition applications (see survey [Tou80b]). Depending on the application, the convex hull may either give too detailed a shape description or may not give enough. Edelsbrunner *et al* [EKS83, EM94] overcame this lacuna by a natural generalization of convex hull to the notion of **α-shapes**. Here α is a real-valued parameter; so we get a spectrum of shapes by varying α.

The α-shape of a point set is defined in terms of generalized discs. An α-disc, for an arbitrary real value α, is defined as follows:

- if $\alpha > 0$, it is a closed disc of radius $1/\alpha$;
- if $\alpha = 0$, it is a closed half-plane; and
- if $\alpha < 0$, it is the closed complement of a disc of radius $-1/\alpha$.

For a given α, the α-hull of a point set P is the intersection of all α-discs that contain P. For $\alpha = 0$, the α-hull is exactly the convex hull. An α-shape of a point set is the boundary of the α-hull, with curved edges being replaced by straight edges. As shown in Fig. 1, a large value of α tends to produce a crude shape of the points, whereas a smaller value of α gives a more detailed shape.

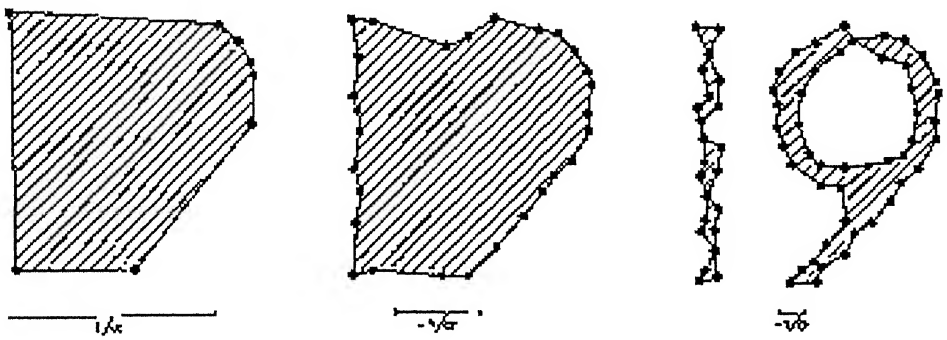


Figure 1: α-shape for various valuesm of α. (Taken from [Ede87].)

The notion of α-shapes has been further generalized to **weighted α-shapes** [Ede92, VBW94].

An α-shape is inadequate for a point set of variable density. For example, when we have two clusters of different densities. Melkemi [Mel97] proposed the notion of **$\mathcal{A}$-shapes** to address this situation. The $\mathcal{A}$-shape of a set of points P is defined by means of a set $\mathcal{A}$ of control points as follows.

Let $R(\mathcal{A}, p_i)$ be the Voronoi region of $p_i \in P \cup \mathcal{A}$. Let $\mathcal{A}'$ denote the extremities of the Voronoi edges that separate points in set P from those in $\mathcal{A}$ in the Voronoi diagram $Vor(P \cup \mathcal{A})$. For q in $\mathcal{A}'$, let $Cdisk(q)$ be the closed complement of the Delaunay disc centered at q.

The $\mathcal{A}$-hull of P, $\mathcal{H}(\mathcal{A}, P) = [\cap_{q \in \mathcal{A}'} Cdisk(q)] \cap B(\mathcal{A}, P)$, where $B(\mathcal{A}, P) = \cup_{p \in S} R(\mathcal{A}, p)$

A Delaunay edge $\overline{p_i p_j} \in DT(P \cup \mathcal{A})$ is an $\mathcal{A}$-edge if there exists a Delaunay circle passing through p_i, p_j, and a point in $\mathcal{A}$. The $\mathcal{A}$-shape of P is the set of $\mathcal{A}$-edges. See Fig. 2. The time required to compute an $\mathcal{A}$-shape is in $O(n \log n)$ [Mel97].

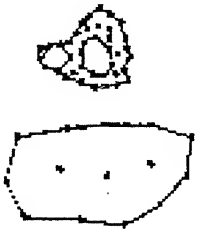


Figure 2: $\mathcal{A}$-shape of a set of points. (Taken from [Mel97].)

1.2 Internal Shapes

> *The internal shape of a set of points is the shape exhibited by identifying the "essential" internal points of the set and, among these, joining "essential" neighbors* [KR85].

The most elementary internal shape descriptor is obtained by joining all **closest pairs** of points. The closest pairs of points give an approximate density of the point set. This can be computed in $\theta(n \log n)$ time [PS85]. Another elementary shape descriptor is the graph obtained by joining each point to all its nearest neighbors. This graph is called the Nearest Neighbor Graph (NNG). The NNG of a point set can be computed in $O(n \log n)$ time [PS85].

At the next level of complexity, some well-known internal shape descriptors are: **The Sphere of Influence Graph** (SIG) [Tou88], **The Relative Neighborhood Graph** (RNG) [Tou80c], **The Gabriel Graph** (GG) [MS84], and **The Delaunay Triangulation** (DT) [Del34]. We discuss these in detail below.

For each point p_i, let r_i be the distance to a closest point. The circle of influence of p_i is the circle of radius r_i, centered at p_i. The Sphere-of-Influence graph is obtained by joining each pair of points p_i and p_j whose circles of influence intersect in more than one point. This can be computed in $O(n \log n)$ time [Tou88].

Two points p_i and p_j are relative neighbors if and only if the following inequality holds. $d(p_i, p_j) \leq max\{d(p_i, p_k), d(p_j, p_k) | p_k \in P - \{p_i, p_j\}\}$. Intuitively, this means that there is no point that is closer to both p_i and p_j than they are from each other. The Relative Neighborhood Graph (RNG) is a geometric graph obtained by joining each pair of points that are relative neighbors. The RNG can be computed in $O(n \log n)$ time [Sup83, Hwa90] The RNG is a subgraph of the Delaunay triangulation (DT). Given the DT of a point set, the RNG can be computed in linear time [Lin94, JK87].

Two points p_i and p_j are Gabriel neighbors if and only if the circle of radius $d(p_i, p_j)/2$ passing through the points p_i and p_j is empty. The graph obtained by joining all Gabriel neighbors is called the Gabriel Graph (GG). It can be computed in $O(n \log n)$ time [MS84]. The GG can be computed in linear time if the Delaunay triangulation of the point set is given [JT92].

The Delaunay Triangulation (DT) is a maximal planar descriptor of internal structure. It has several interesting properties [PS85]. Two points p_i and p_j are Delaunay neighbors if there exists an empty circle passing through them. The Delaunay triangulation is a graph obtained by joining all Delaunay neighbors. The scope of applications of this geometric graph embrace fields as diverse as biology, crystallography, ecology, to name a few. It can be computed in $O(n \log n)$ time [PS85].

Just as the α-shapes give rise to a continuous spectrum of external shapes by varying the parameter α, β-skeletons, introduced by [KR85], give rise to a continuous spectrum of internal shapes by varying the real parameter β. This can be used to extract the boundary of an objects [ABE97].

More specifically, a β-skeleton is a geometric graph, obtained by joining pairs of points whose β-neighborhoods are empty. The neighborhood size depends on the parameter β, and are of two types.

The circle-based β-neighborhood of a pair of points (p_i, p_j) is defined as follows: for $\beta \geq 1$, it is the union of all open circles of radius $\beta|\overline{p_i p_j}|/2$, passing through p_i and p_j. For $\beta \in [0, 1]$, it is the **interior** of the intersection of two open circles of radius $|\overline{p_i p_j}|/(2\beta)$, passing through p_i and p_j. See Fig. 3. For $\beta = 1$ is exactly equal to the Gabriel graph. Given the DT of the point set, the circle based β-skeleton can be computed in linear time [KR85], for $\beta \geq 1$,

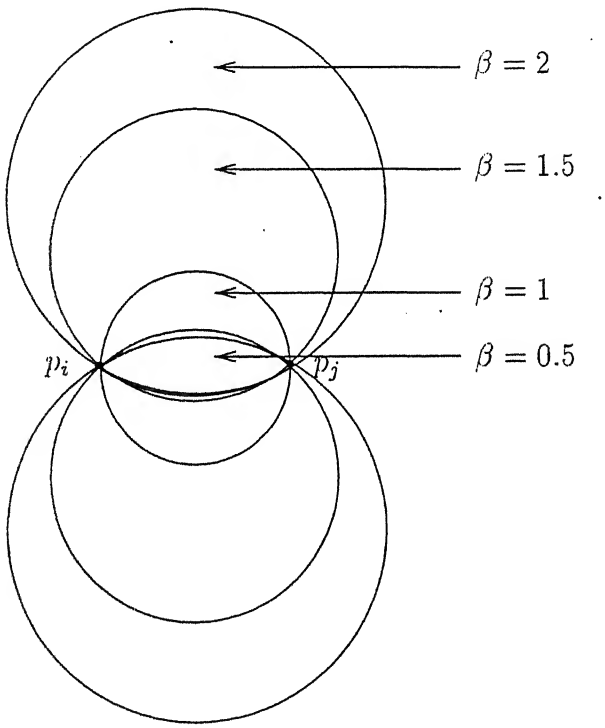


Figure 3: Circle-based β-neighborhood(p_i, p_j) for various $\beta > 0$.

The lune-based β-neighborhood of a pair (p_i, p_j) of points is defined as follows: for $\beta \geq 1$, it is the **interior** of the intersection of two circles of radius $\beta|\overline{p_i p_j}|/2$, centered at the points $(1 - \beta/2)p_i + (\beta/2)p_j$ and $(\beta/2)p_i + (1 - \beta/2)p_j$, respectively. For $\beta \in [0, 1]$, the lune-based neighborhood is identical with the circle-based neighborhood. See Fig. 4.

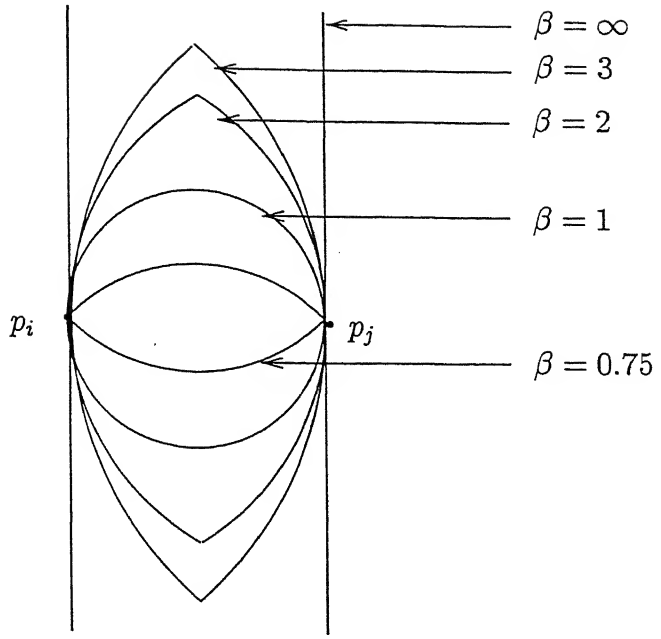


Figure 4: Lune-based β-neighborhood(p_i, p_j) for various $\beta > 0$.

For $\beta = 1$ is exactly equal to the Gabriel graph and for $\beta = 2$ it coincides with the RNG. The β-skeleton for $1 \leq \beta \leq 2$ can be computed from the DT in linear time [Lin94, JKY] in the metric L_p, provided $p = 2$ or $\beta = 2$.

1.2.1 Other Shape Descriptors

Another important shape descriptor which captures internal as well as external shape of a point set is the γ-skeleton [Vel92b, Vel92a, Vel94]. It is defined in terms of two parameters c_0 and c_1, where $c_0, c_1 \in [-1, 1]$ and $|c_0| \leq |c_1|$. The γ-neighborhood of a pair of points p_i and p_j is defined by two circles of radius $d(p_i, p_j)/2(1 - |c_0|)$, and $d(p_i, p_j)/2(1 - |c_1|)$ passing through p_i and p_j such that

- if $c_0 c_1 < 0$: the centers of the circles lie on the same side of the line passing through p_i, p_j,

- if $c_0c_1 > 0$: the centers of the circles lie on opposite sides of the line passing through p_i, p_j,
- if $c_1 \in [-1, 0]$: the γ-neighborhood is the interior of the intersection of these circles, plus the interior of the edge connecting c_0 with c_1,
- if $c_1 \in [0, 1]$: the γ-neighborhood is the interior of the union of these circles, plus the interior of the edge connecting c_0 with c_1,

For given points p_i, p_j and c_0, c_1 satisfying the above conditions, there are two γ-neighborhoods. Two points are γ-neighbors if and only if at least one of the γ-neighborhood is empty. The γ-skeleton is a graph obtained by joining all the γ-neighbors. See Fig. 5.

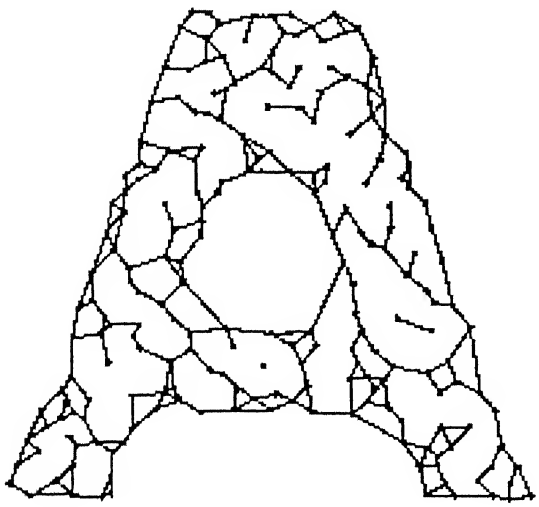


Figure 5: $\gamma(-0.15, 0.3)$-skeleton of a set of points. (Taken from [Vel94].)

1.3 Organization of Thesis

In this thesis we study some of the algorithmic aspects of lune-based β-skeletons. In the next chapter, we discuss a sweepline algorithm for constructing a β-skeleton

for any $\beta \geq 0$. The time-complexity of our algorithms is in $O(n^{3/2} \log n)$ for $\beta \geq 1$ and in $O(n^{5/2} \log n)$, for $\beta \in [0, 1)$ under the metric L_p for $1 < p < \infty$. In the metrics L_1 and L_∞, our algorithm takes $O(n^{5/2} \log n)$ time for any $\beta \geq 1$. The concept of a β-skeleton have natural generalizations to that of $k\beta$-skeletons and additively weighted β-skeletons. We have shown that the above sweepline method can be extended to construct these geometric graphs too. In chapter 3 we present two output-sensitive algorithms for computing a β-skeleton for $\beta \geq 2$, in the metries L_1 and L_∞.

In the fourth chapter we present efficient algorithms for computing the entire spectrum of β-shapes. Our algorithms are in $O(n^{3/2} \log n + K)$ for $\beta \geq 1$ and in $O(n^2 \log n + K)$ time for $\beta \geq 0$ in L_2 metric. A point that lies in the β-neighborhood of an edge is a witness to this edge. The quantity K is a count of the number of times that the edges of the initial graph on P are witnessed for $\beta = \infty$. We also present algorithms for computing the spectra of of $k\beta$-shapes and additively-weighted β-shapes.

In chapter 5, we extend the definition of β-skeleton to higher-dimensions. We show that the size of a β-skeleton for $\beta > 2$ in higher dimensions is linear. We also present an efficient algorithm for computing a β-skeleton for $\beta > 2$.

We conclude and discuss directions for further research in the last chapter.

Chapter 2

Computing β-Skeletons and Their Relatives

2.1 Introduction

In this chapter, we present a sweepline algorithm [PS85] for constructing a β-skeleton for any $\beta \geq 0$. The time-complexity of our algorithm is in $O(n^{3/2} \log n)$ for $\beta \geq 1$ and in $O(n^{5/2} \log n)$, for $\beta \in [0, 1)$ under the metric L_p for $1 < p < \infty$. In the metrics L_1 and L_∞, our algorithm takes $O(n^{5/2} \log n)$ time for any $\beta \geq 0$.

The concept of a β-skeleton have natural generalizations to that of $k\beta$-skeletons and additively weighted β-skeletons. We show that the above method can be extended to construct these geometric graphs too.

The above algorithms are based on an efficient solution to the following **n-circle problem**: Given a set C of n circles and a set P of n points, report all the circles in C that are empty.

This chapter is organized as follows. In the next section, we present an efficient

algorithm for the n-circle problem. In the third section we show how this can be used to compute a β-skeleton for any $\beta \geq 0$. In the following sections, we define the notions of a $k\beta$-skeleton and additively weighted β-skeleton. Algorithms for computing these are presented in the fourth and fifth sections respectively; we conclude in the sixth section.

2.2 The n-Circle Problem

2.2.1 Preliminaries

The n-circle problem we stated in the introduction can be solved as follows: Compute the Voronoi diagram of the query points. Let v_i denote the Voronoi region of a point p_i. For each circle c_j, we find the Voronoi region v_i that contains the center of c_j. The circle c_j is non-empty, if the distance between p_i and its center is less than its radius, that is $p_i \in c_j$. The time complexity of this algorithm is in $O(n \log n)$. But the problem with this method is that checking the emptiness of lunes is no longer as simple.

Another method of solving the n-circle problem is by repeated point location query in an arrangement of n circles. We could use Bentley and Maurer's algorithm for this purpose [BM79], which preprocesses the circles in $O(n^3)$ time for point location. It takes a query point, and outputs the circles that contains this point in $O(\log\ n + K)$ time, where K is size of the output.

For the problem at hand, however, we know all the query points off-line. Can we make use of this fact to solve our problem more efficiently? The following observations show that the answer is yes.

Observation 2.1 *An arrangement of n circles divides the plane into a set R of at most $O(n^2)$ regions.*

Proof: Let $n-1$ circles divide the plane into $T(n-1)$ regions. The n^{th} circle intersects at most $n-1$ circles. These intersection points divide the boundary of the n^{th} circle into at most $2(n-1)$ disjoint arcs, since there are at most $2(n-1)$ intersection points and two consecutive intersection points define an arc. Each arc of the n^{th} circle divides a region into two. So the extra regions created by the n^{th} circle is at most $2(n-1)$. Therefore $T(n) = T(n-1) + O(2n-2)$. This implies that $T(n) = O(n^2)$, since T(1)= 1. ∎

Observation 2.2 *If a circle is not empty, then it is sufficient to find one witness (that is, a point lying inside the circle) for it.*

Observation 2.3 *If a point is in the region $r \in R$, then all the circles that contain this region are non-empty and can be pruned.*

2.2.2 An Algorithm

The main idea of our algorithm is that instead of pre-computing the arrangement and then doing repeated point location, we can achieve the same effect more efficiently by sweeping the circles and the points simultaneously. For details on the sweepline paradigm we refer the reader to [PS85].

We shall use the following notations and definitions throughout this chapter. For each position of the sweepline, we define a partial order relation $<_a$ in the set of regions R as follows. For two regions r_i and r_j in R, $r_i <_a r_j$ if the intersection of the region r_i with vertical line $x = a$ lies below the intersection of the region r_j with that line. This ordering information is maintained in the *sweepline status* $\mathcal{L}$. An *event point* is either an intersection point of two circles, or the leftmost/rightmost point of a circle, or an input point. The events are maintained in a priority queue Q, which is initialized to contain the input points and the left-end points of the circles. A circle in C is said to be *active* if it is either empty or a witness is yet to be found.

We denote the set of active circles by A_c. Initially, $A_c = C$. For each region $r \in \mathcal{L}$, let C_r denote the set of active circles that contain this region.

To describe the algorithm it is enough to specify the actions at each type of event point as the sweep proceeds.

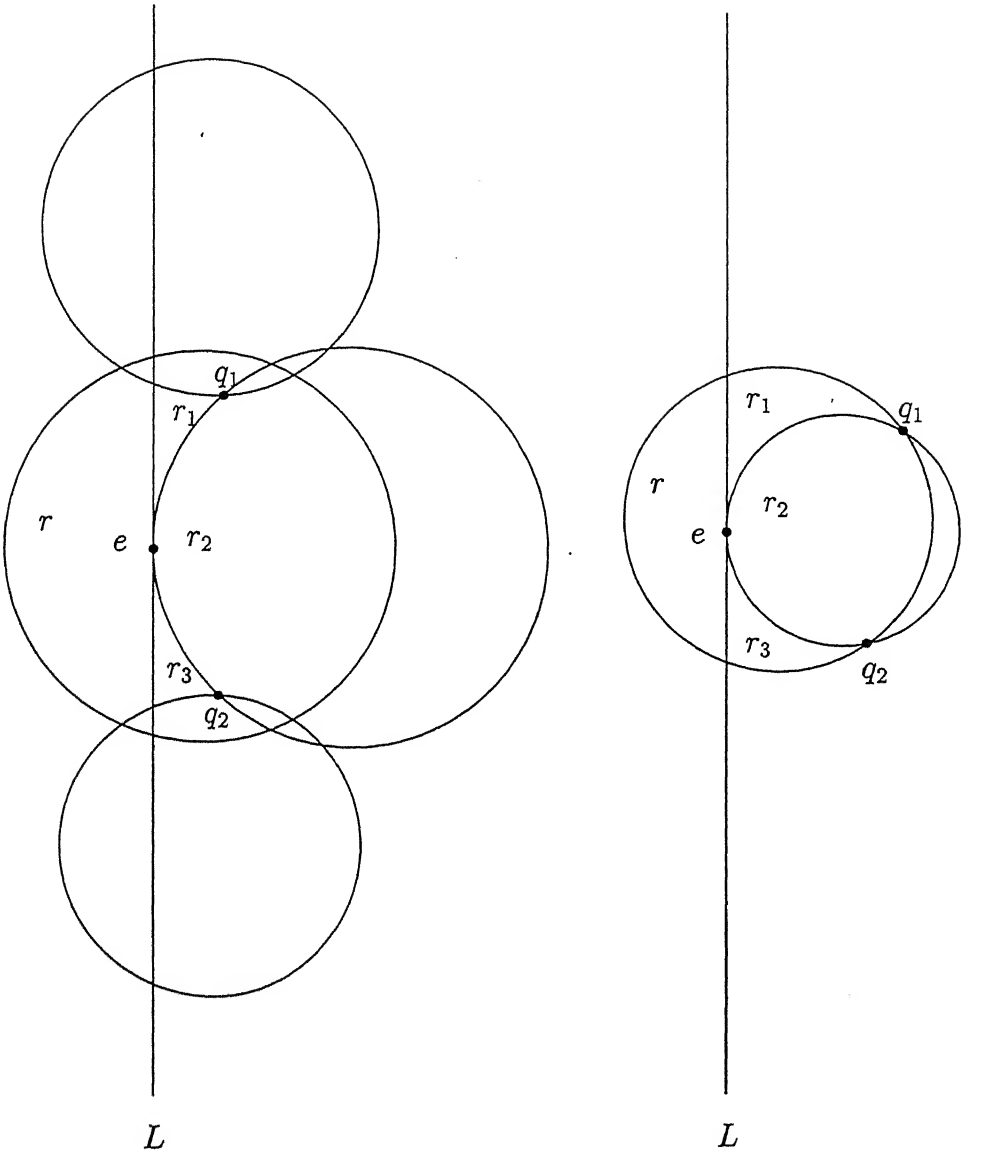
Let the event e correspond to the *left end point* of a circle c. We insert the right-end point of the circle in Q and locate the region $r \in \mathcal{L}$ that contains this event point. The boundary of c splits r into three regions r_1, r_2, and r_3, as shown in Fig. 6.

Comment : *It may be noted in passing that r_1 and r_3 are the same region r as can also be seen from the Fig. 6; however, we distinguish between them because in the sweepline status a region is identified and maintained by the two circular arcs that bound it from above and below. Thus r appears in two incarnations in the sweepline status in this case.* ∎

Therefore, we replace the region r by the regions r_1, r_2, and r_3 in $\mathcal{L}$. Simultaneously, we create the active circle lists C_{r_1}, C_{r_2}, and C_{r_3}; the first and third are really copies of C_r, while the second is obtained from C_r by adding to it the circle just encountered. If c intersects the circular arc that bounds r_1 from above we insert this intersection point in Q. Similarly, for the circular arc that bounds r_3 from below. These intersection points are q_1 and q_2 in Fig. 6.

Let the event e corresponds to the *right end point* of a circle c. If this point is on the boundary of the regions r_1, r_2, and r_3, then the region r_2 is deleted from $\mathcal{L}$ and the regions r_1 and r_3 are coalesced into a new region r_{13} (see Fig. 7). The active circle list $C_{r_{13}}$ of this new region is the same as that of r_1 or r_3. We insert into the event queue the intersection point, if any, of the bounding arcs of the region r_{13} (point q in Fig. 7). This intersection point can very well be the right-end point of a circle as shown in Fig. 8.

Let the event e correspond to an *intersection point* of two circles c_1 and c_2. If e



(a) (b)

Figure 6: Update of $\mathcal{L}$ at the left-end point of a circle.

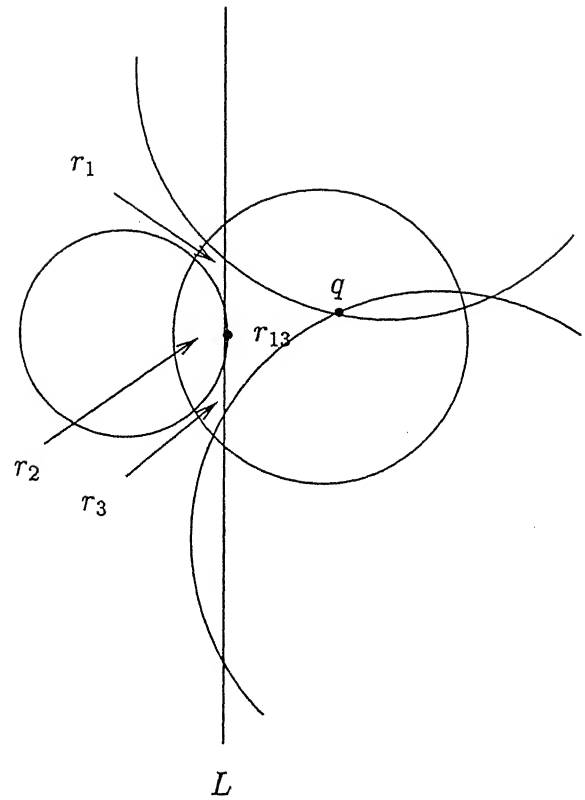


Figure 7: Update of $\mathcal{L}$ at the right-end point of a circle.

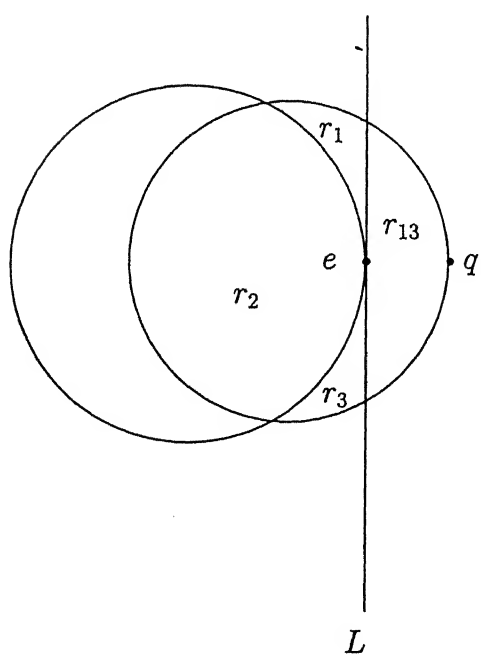


Figure 8: Update of $\mathcal{L}$ at the right-end point of a circle.

is on the boundary of the regions r_1, r_2, and r_3, then these regions transform into new regions r_1', r_2' and r_3' respectively. The region r_1' has a new lower boundary vis-a-vis r_1 and the same active circle list; r_2' has new upper and lower boundaries as compared to r_2 and an active circle list obtained from C_{r_2} by deleting/adding the intersecting circles c_1 and c_2; r_3' has a new upper boundary compared with r_3 and the same active circle list. If the bounding arcs of r_1' intersect, we insert their intersection point in Q. We do the same for r_3'. In Fig. 9 these intersection points are q_1 and q_2. Notice that the latter is the right end point of one of the intersecting circles.

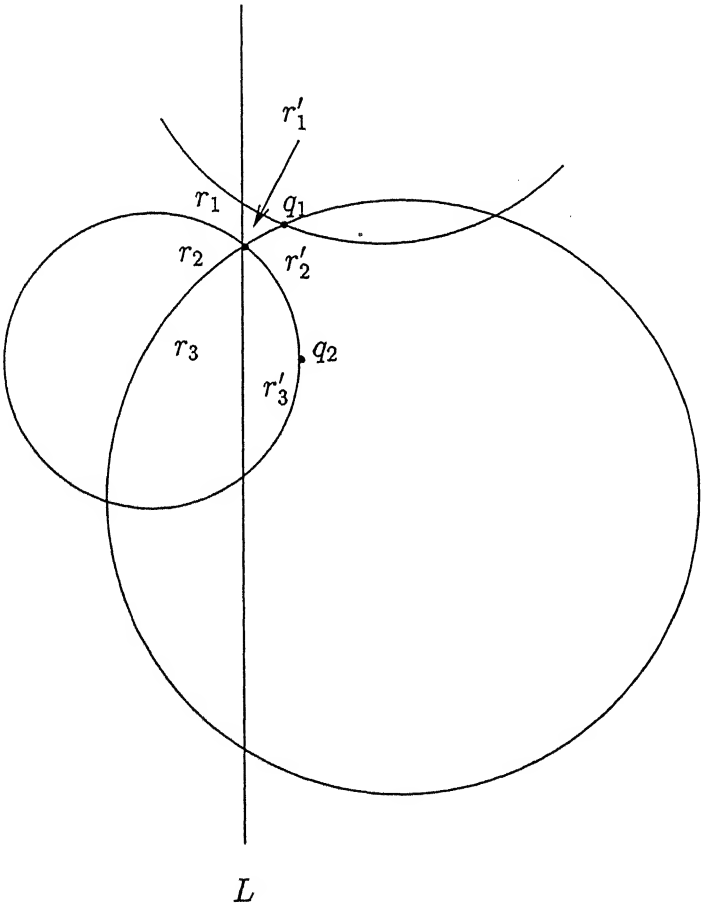


Figure 9: Update of $\mathcal{L}$ at an intersection point of two circles.

Let the event e correspond to an *input point*, we locate the region $r \in \mathcal{L}$ that contains this point and remove all the circles in C_r (see Fig. 10). The removal of

each such circle involves deleting it from the set of active circles A_c, deleting the intersection points of circle c with other circles, which are present in Q, and updating $\mathcal{L}$.

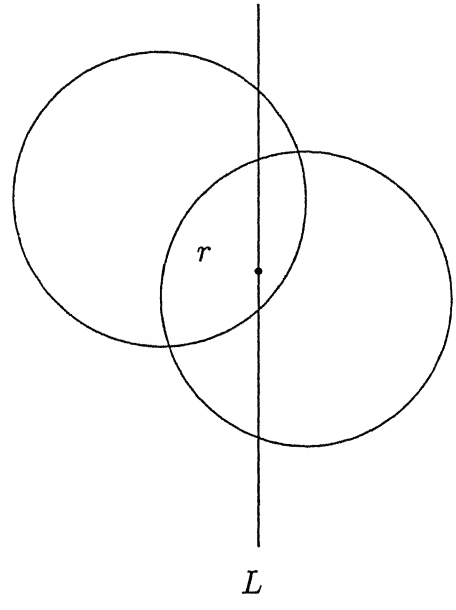


Figure 10: Update of $\mathcal{L}$ at an input point.

2.2.3 Analysis of the Algorithm

We claim the following result.

Lemma 2.1 *The n-circle problem can be solved in $O(n^2 \log n)$ time, using $O(n^2)$ space.*

Proof: At any abscissa, the sweepline has at most $2n$ intersections with the circles; hence at most $2n + 1$ segments. Since each segment corresponds to a region, the sweepline status can have at most $O(n)$ regions.

Any data structure used for implementing the sweepline status $\mathcal{L}$ should support efficient implementations of the following operations: addition and deletion of a

region, point location, and merging of two adjacent regions. A height-balanced tree meets these needs eminently. The regions are stored at the leaves and the internal nodes correspond to the circular arcs that make-up the region boundaries. The active circle list C_r of each region r in the sweepline status is also maintained as a height-balanced tree.

We maintain a Boolean array $Prune[]$ to keep track of the status (empty/non-empty) of all the circles. Initially, $Prune[i]$ is $false$ for each i in $1 \leq i \leq n$. During the sweep, if circle c_i is found to be non-empty we set $Prune[i]$ to $true$.

The event queue, as well as the array $Prune[n]$, can be initialized in $O(n)$ time.

We analyze separately the complexity of processing each of the four types of events in Q.

Let e be an event that is the left end point of a circle c. The cost of locating this point in $\mathcal{L}$ is in $O(\log n)$. If r be the region that contains this event point, we replace it by three new regions r_1, r_2, and r_3 in $\mathcal{L}$ and if the bounding arcs of the region r_1 intersect, insert their intersection point in Q. We do the same for the region r_3. The complexity of this is in $O(\log n)$ again. Creating the active circle lists corresponding to these regions is in $O(n)$, since we have to make three copies of C_r and insert the newly encountered circle into one of these. The cost of processing the left-end point of a circle is in $O(n)$ time. The total cost of processing the left-end points of all the circles is in $O(n^2)$ time.

A similar analysis shows that the cost of processing the right-end point of a circle is in $O(\log n)$. The total cost of processing the right-end points of all the circles is in $O(n \log n)$.

Let the event e be an intersection point. The cost of locating this point in $\mathcal{L}$ and hence the three consecutive regions, r_1, r_2, and r_3, that have e is on their boundary is in $O(\log n)$. The cost of replacing these regions in $\mathcal{L}$ by the corresponding regions r'_1, r'_2 and r'_3 is also in $O(\log n)$. The cost of creating the active circle lists corresponding

to these regions is in $O(\log n)$, since we only need to make two deletions/insertions into the list corresponding to r_2 to get the active circle list of r_2'. The active circle lists of r_1' and r_3' are same as that of r_1 and r_3 respectively. We insert in Q, the intersection points of the arcs that bound the region r_1' as discussed earlier. We do the same for r_3'. The complexity of this is in $O(\log n)$ again. So the cost of processing one intersection point is in $O(\log n)$ time. The cost, therefore, of processing $O(n^2)$ intersection points is in $O(n^2 \log n)$.

Let the event e be an input point. The complexity of locating this point in the sweepline status is in $O(\log n)$. If r be the region that includes this point, we delete all the circles from its active list C_r and set the appropriate array entries in $Prune[]$ to *true*.

If the point e is on the boundary of the circle c and one of the bounding arc of the region r is this circle c. Let the regions r_1 and r_2 are respectively above and below the point e in the sweepline status $\mathcal{L}$ (see Fig. 11). The circle lists of these regions are differ by the circle c. That is, $C_{r_1} - \{c\} = C_{r_2}$ or $C_{r_2} - \{c\} = C_{r_1}$. The point e is not a witness for c, but it is a witness of all other circles of the list C_{r_1} and C_{r_2}. Therefore, we prune all the circles in $C_{r_1} - \{c\}$.

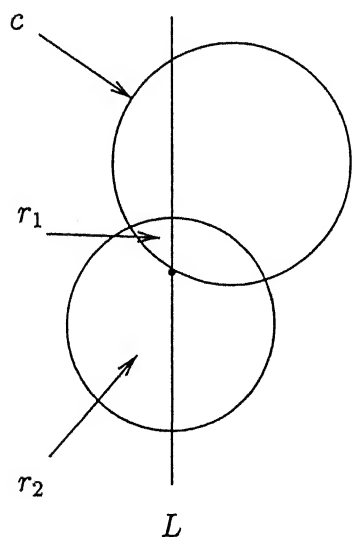


Figure 11: Update of $\mathcal{L}$ at an input point, if it is on the boundary of a region.

The cost of deleting a circle c_i is in $O(n \log n)$. We argue as follows. As observed earlier, this involves removing from the event queue its intersections with other circles that are present in Q as also its right end-point; it also involves removing this circle from the active list of all the regions in $\mathcal{L}$ in which it occurs. The first can be done in constant time by the following artifice. We merely set $Prune[i]$ to *true*. While processing an intersection point q or a right end point we have to take the additional care of checking suitable array entries in the array $Prune[]$ to decide if q needs to be processed. The cost of the second is in $O(n \log n)$. The adjustments required to $\mathcal{L}$ is in $O(\log n)$ since the removal of its upper/lower arc causes the regions above and below this arc to be merged into one. In the course of the sweep at most n circles are pruned, and thus the amortized cost is in $O(n^2 \log n)$.

Hence the n-circle problem can be solved in $O(n^2 \log n)$ time using $O(n^2)$ space. ∎

The complexity of the above algorithm is worse than the brute-force algorithm which tests a point for inclusion in each of the n circles! For the complexity of this is in $O(n)$, and therefore in $O(n^2)$ for n points. However, if we look at the above algorithm very carefully we notice that the problem lies in considering all the circles in one go. The processing time for the n circles is high and dominates the algorithm. The following lemma shows how to balance the processing time for circles and points to improve the time complexity.

Lemma 2.2 *Given a set C of n circles and a set P of n points, all the empty circles can be computed in $O(n^{3/2} \log n)$ time using $O(n)$ space.*

Proof: We divide the problem into n/m subproblems, each subproblem consisting of m circles and all n input points. For each of these subproblems the above algorithm takes $O(m^2 \log m + n \log m)$ time. Therefore, the total time required to solve all the n/m problems is $O(nm \log m + n^2/m \log m)$ time. The minimum

time of $O(n^{3/2} \log n)$ is achieved for $m = \sqrt{n}$. The space required is in $O(n)$, since each subproblem consists of $\sqrt{n}$ circles. Hence the claim of the lemma. ∎

2.3 β-Skeleton

In this section, we generalize the above algorithm to compute the β-skeleton for $\beta \geq 1$.

It is well-known that the Delaunay triangulation (DT) [PS85], is a super-graph of a β-skeleton for $\beta \geq 1$. Therefore, to compute the β-skeleton, for some $\beta \geq 1$, we construct DT(P) and prune the edges whose β-lunes are not empty. The Delaunay triangulation can be constructed by any standard algorithm in $O(n \log n)$ time. What remains is to find a way to prune the non-empty lunes.

It is easy to extend the solution to the n-circle problem, if circles are replaced by lunes. The same observations have been made by Su and Chang in [SC91b]. In fact, we can consider any set of convex objects with constant-size description, any two of which have a bounded number of intersections. With lunes, the event structure is slightly more complicated. Unlike in the case of circles, each lune can have up to four events associated with it. Two of these correspond to positions where a lune joins the sweepline and leaves it. The other two correspond to the intersections of the circles that define a lune (i_1 and i_2 in Fig. 12).

Due to this event structure the region in the sweepline status that reflects its membership in $\mathcal{L}$ can be defined by three different pairs of arcs. For example in Fig. 12 these are the pair $\{a, b\}$, $\{a, c\}$ and $\{c, d\}$ consecutively. The rest of the details carry over and we have the following claim:

Theorem 2.3 *For $\beta \geq 1$, the β-skeleton of P can be constructed in $O(n^{3/2} \log n)$ time using $O(n)$ space.*

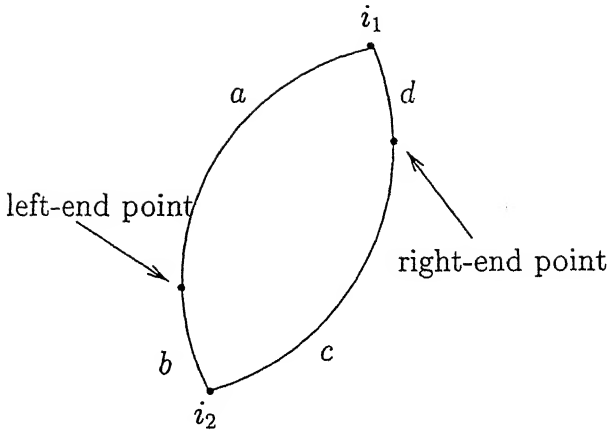


Figure 12: Initial events of a lune.

Similarly, the β-skeleton of P, for $\beta \in [0, 1)$, can be constructed by starting with the complete graph on P as the super graph and pruning the non-empty lunes. The following theorem can be proved along exactly the same lines

Theorem 2.4 *The β-skeleton of a set P of n points in the plane, for $\beta \in [0, 1)$, can be constructed in $O(n^{5/2} \log n)$ time using $O(n^2)$ space.*

Proof: We divide $O(n^2)$ lunes corresponding to the edges in the complete graph into n^2/m subproblems. Each subproblem consists of m lunes and all n input points. For each of these subproblems the above algorithm takes $O(m^2 \log m + n \log m)$ time. Therefore, the total time required to solve all the n^2/m problems is $O(n^2 m \log m + n^3/m \log m)$ time. The minimum time of $O(n^{5/2} \log n)$ is achieved for $m = \sqrt{n}$. The space required is in $O(n^2)$, since we are start with complete graph. Hence the claim of the lemma. ∎

2.4 $k\beta$-Skeleton

We can generalize the notion of a β-skeleton by relaxing the emptiness criterion and allow the interior of a lune to contain up to $k - 1$ points; we get what is called a

$k\beta$-skeleton. The geometric graphs that we get for different values of β, 1 upwards, are all subgraphs of what is called the k-Delaunay graph.

The k-Delaunay graph is a generalization of the Delaunay triangulation on n points. The triangulation in this case can be characterized this way: a triangle belongs to the k-Delaunay graph if its circumcircle contains at most $k-1$ points. We can obtain it from the kth-order Voronoi diagram as follows: an edge $\overline{p_i p_j}$ belongs to the k-Delaunay graph if a segment of the bisector of p_i and p_j is in the kth-order Voronoi diagram [SC90].

Algorithms for computing $k\beta$-skeletons for special values of β are extant in the literature. Su and Chang [SC90] has shown how to compute the k-Gabriel graph ($\beta = 1$) in $O(k^2\, n \log n)$ time. For fixed values of k, Chang *et al.* [CTL92b] presented an $O(n^2)$ algorithm to compute the k-Relative Neighborhood Graph ($\beta = 2$). Later, Su *et al.* improved this to $O(n^{5/3} \log n)$ [SC91b]. An interesting application of the k-RNG has been made to the *Euclidean bottleneck matching problem*, by Chang *et al* [CTL92b]. In this problem a matching has to be found among n points in the Euclidean plane that minimizes the longest edge.

To compute a $k\beta$-skeleton for $\beta \geq 1$, we start with the k-Delaunay graph of P and prune those edges whose lunes contain more than $k-1$ points. It turns out that the above n-circle algorithm can be adapted to this case also. The additional work involved is to maintain a counter for the number of points inside each lune. We increment this count for the appropriate set of lunes whenever an input point is encountered during the sweep. Whenever the counter value of any lune becomes k, we prune that lune from further processing. The extra time required to maintain all the counters is in $O(k^2\, n)$, since there are $O(k(n-k))$ counters (the k-Delaunay graph has $O(k(n-k))$ edges [SC90]) and each counter is incremented at most k times. We have thus proved the following theorem.

Theorem 2.5 *The $k\beta$-skeleton of a set P of n points in the plane for $\beta \geq 1$ can be constructed in $O(kn^{3/2} \log n + k^2\, n)$ time.*

It is quite easy to compute a $k\beta$-skeleton, for values of β in the range [0,1). We need to start with the complete graph on P, and prune edges which are not in the $k\beta$-skeleton. The additional cost of this is reflected in the following result.

Theorem 2.6 *The $k\beta$-skeleton of a set P of n points in the plane, for $\beta \in [0,1)$ can be constructed in $O(n^{5/2} \log n + n^2 k)$ time.*

2.5 Weighted Neighborhood Graphs

In this section, we discuss various additively weighted neighborhood graphs. We assign a positive weight w_i to each point $p_i \in P$. The weight w_i expresses the power of the point p_i in the sense that each weighted point p_i can be treated as a circle B_i, centered at p_i, with radius equal to the weight w_i. Thus the weighted distance $d_w(p_i, x)$, between a point $p_i \in P$ and an arbitrary point x, is $d(p_i, x) - w_i$ if x is outside B_i, and zero otherwise. The weighted distance $d_w(p_i, p_j)$, between any two points $p_i, p_j \in P$ is the distance between B_i and B_j. Let D denote the set of circles B_i. We discuss the details of a few individual special cases

2.5.1 Weighted Gabriel Graph (WGG)

The **weighted Gabriel neighborhood**, $N_G(p_i, p_j)$, of two points $p_i, p_j \in P$ that have disjoint circles B_i and B_j is an open circle of radius $d_w(p_i, p_j)/2$, whose center is on the line segment $\overline{p_i p_j}$ and is tangent to the circles B_i and B_j (see Fig. 13). If B_i and B_j intersect, then $N_G(p_i, p_j)$ is empty.

The **weighted Gabriel graph** of a set of points P is a geometric graph (P, E) such that for every $p_i, p_j \in P$, the edge $\overline{p_i p_j} \in E$ if and only if $N_G(p_i, p_j)$ is not intersected by any circle $B_k \in D - \{B_i, B_j\}$.

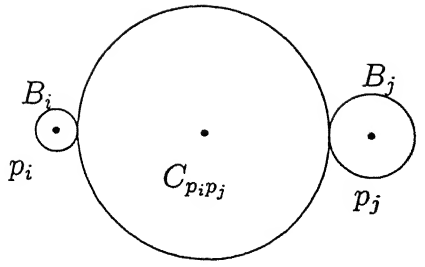


Figure 13: Weighted Gabriel neighborhood.

The **weighted Delaunay graph** is a dual of the weighted Voronoi diagram [AE84] and can be computed in $O(n \log n)$ time [Mir93]. An edge $\overline{p_ip_j}$ belongs to this graph, if there is a common boundary between the Voronoi regions of p_i and p_j. The following result and its proof are from Mirzaian [Mir93].

Lemma 2.7 *The weighted Gabriel graph is a subgraph of the weighted Delaunay graph if the set of circles D is pairwise disjoint.*

Proof: Let the edge $\overline{p_ip_j} \in$ WGG. The Gabriel neighborhood of p_i and p_j, $N_G(p_i, p_j)$, is not intersected by any circle in D. Let the point $C_{p_ip_j}$ be the center of the Gabriel Neighborhood $N_G(p_i, p_j)$. The circles B_i and B_j are at an equal distance from $C_{p_ip_j}$. Moreover, no circle $B_k \in D - \{B_i, B_j\}$ is closer to $C_{p_ip_j}$ than the circles B_i and B_j. From the definition of a Voronoi diagram, the point $C_{p_ip_j}$ is on the bisector of the sites B_i and B_j. Thus by definition $\overline{p_ip_j} \in$ WDG [Mir93]. ∎

2.5.2 Weighted Relative Neighborhood Graph (WRNG)

The **weighted relative neighborhood**, $N_{RNG}(p_i, p_j)$, of two points $p_i, p_j \in P$ that have disjoint circles B_i and B_j is the interior of the intersection of two circles, of radii $d_w(p_i, p_j) + w_i$ and $d_w(p_i, p_j) + w_j$, centered respectively at the points p_i and p_j (see Fig. 14). If B_i and B_j intersect then $N_{RNG}(p_i, p_j)$ is empty.

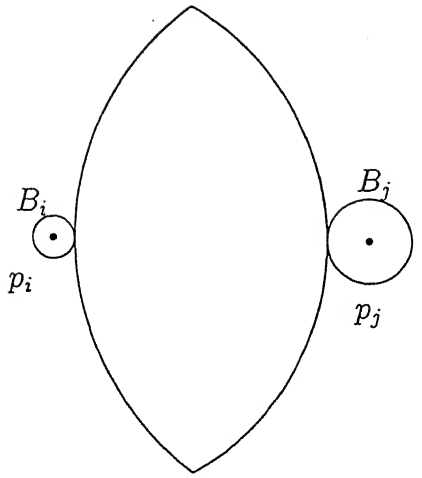


Figure 14: Weighted relative neighborhood.

The weighted relative neighborhood graph of P is defined as follows. For every $p_i, p_j \in P$, the edge $\overline{p_i p_j}$ is in the graph if and only if the following inequality holds. $d_w(p_i, p_j) \leq max\{d_w(p_i, p_k), d_w(p_j, p_k)\}$, for all $p_k \in P - \{p_i, p_j\}$ (see Fig. 14). Or equivalently, if the lune is not intersected by any circle in $D - \{B_i, B_j\}$.

Mirzaian [Mir93] made a very interesting application of the weighted RNG to the problem of *Minimum Weight Euclidean Matching*. In this problem we are given $2n$ points in the Euclidean plane and are required to match them into n pairs such that the sum of the n distances between the matched pairs is minimum.

2.5.3 Weighted β-Skeleton

For all $p_i, p_j \in P$ such that B_i and B_j are disjoint, the weighted lune-based neighborhood, $lune_\beta(p_i, p_j)$, is a region lying between the circles B_i and B_j and is defined as follows: for $\beta \geq 1$, it is the interior of the intersection of two circles of radii $\beta d_w(p_i, p_j)/2 + w_i(\beta - 1)$ and $\beta d_w(p_i, p_j)/2 + w_j(\beta - 1)$, whose centers are on the line joining p_i and p_j, and which touch externally the circles B_j and B_i respectively (see Fig. 15). For $\beta \in [0, 1]$, it is the interior of the intersection of two circles of radius $d_w(p_i, p_j)/2\beta$, passing through p_i' and p_j' which are a pair of closest points

on the circles B_i and B_j (see Fig. 15). If the circles B_i and B_j intersect, then the $lune_\beta(p_i, p_j)$ is empty for all $\beta \geq 0$.

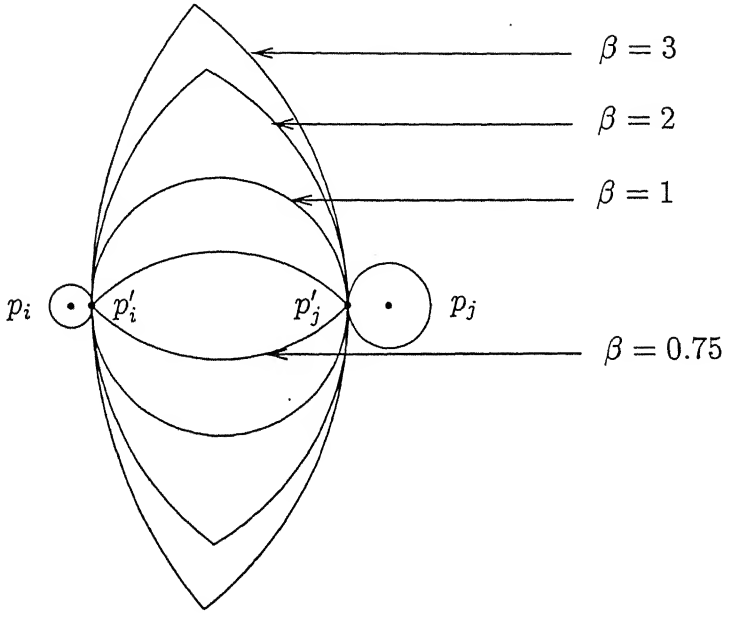


Figure 15: Neighborhood of weighted β-skeleton.

The weighted β-skeleton of P in this case is a geometric graph (P, E) such that for every $p_i, p_j \in P$, the edge $\overline{p_i p_j} \in E$ if and only if $lune_\beta(p_i, p_j)$ is not intersected by any circle $C_k \in D - \{B_i, B_j\}$, where D is the set of circles B_i.

When $\beta = 1$, $lune_\beta(p_i, p_j)$ is exactly the Gabriel neighborhood $N_G(p_i, p_j)$, and when $\beta = 2$, we get the weighted relative neighborhood, $N_{RNG}(p_i, p_j)$..

Lemma 2.8 *For a pairwise disjoint set of circles D and $\beta \geq 1$, the weighted β-skeleton is a subset of the* WDG.

Proof: This lemma follows from the fact that for $\beta = 1$, the weighted β-skeleton is exactly the weighted Gabriel graph, which is a subgraph of WDG from Lemma 2.7.

∎

2.5.4 Algorithm

In this section, we discuss an algorithm for computing a weighted β-skeleton for $\beta \geq 1$. We assume that the circles in D are pairwise disjoint.

As we mentioned in the previous section, the WDG is a super-graph of a $W\beta$-skeleton for $\beta \geq 1$. The WDG can be constructed in $O(n \log n)$ time [Mir93].

So we just have to prune the lunes that intersect any circle $B_i \in D$. For the sake of simplicity, we consider circles instead of lunes. As we observed earlier, the generalization to lunes is pretty straightforward. The problem can be stated with a bit of color as follows:

> Given a set of $O(n)$ blue circles C and a set of n red circles D, find all the blue circles that are not intersected by any red circle.

We follow closely the basic circle-point algorithm to design an algorithm for this case. Most of the details carry over, except when in the course of the sweep we meet a red circle we locate its left-end point in a region of the sweepline status and check for intersection with the circles which correspond to the upper and lower arcs that bound this region. If a bounding arc is intersected, the corresponding circle is removed from the active list of blue circles, and the arc is deleted from the sweepline status. This latter action causes two adjacent region to be merged into a new one and we repeat the above sequence of steps for this region. The process terminates when we get a region whose bounding arcs have no intersection with the red circle. Now we insert the red circle into the sweepline status because there can be yet more intersections with blue circles that have not been swept yet. However, such an intersection can occur only after a blue-blue intersection has been encountered during the sweep (see Fig. 16). So, whenever a red-blue intersection is encountered, prune the blue-circle corresponding to this intersection. A routine complexity analysis now gives the following result.

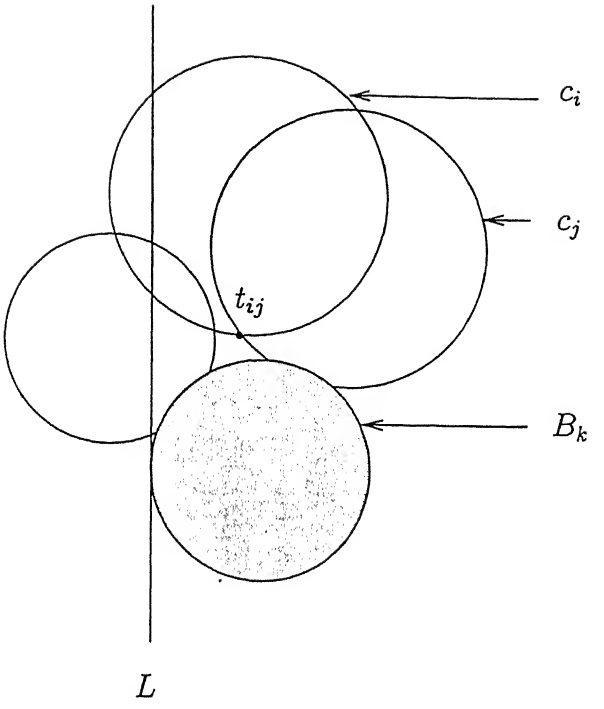


Figure 16: Intersection between the red circle B_k and the blue circle c_j is detected after blue-blue intersection t_{ij}.

Theorem 2.9 *The weighted β-skeleton of a set of n points in the plane for $\beta \geq 1$ can be constructed in $O(n^{3/2} \log n)$ time.*

Theorem 2.10 *The weighted β-skeleton of a set of n points in the plane for $\beta \in [0, 1)$ can be constructed in $O(n^{5/2} \log n)$ time.*

2.6 Conclusion

In this chapter, we have presented an efficient sweepline algorithm for the n-circle problem, and used this to compute β-skeleton, $k\beta$-skeleton, and weighted β-skeleton under the metric L_p for $1 < p < \infty$. In the metrics L_1 and L_∞, the size of these geometric graphs are in $O(n^2)$ for any β. We can compute these graphs in

$O(n^{5/2} \log n)$ time by starting with the complete graph and pruning the non-empty lunes, following the technique of this chapter.

The algorithms presented in this chapter are not optimal. It would be intersecting to find out more efficient algorithms for computing these geometric graphs.

Chapter 3

An Output-Sensitive Algorithm for Computing the β-Skeleton

3.1 Introduction

In the previous chapter we have presented an efficient algorithm for computing the β-skeleton in any metric L_p, for $1 \leq p \leq \infty$. In the metrics L_1 and L_∞, a β-neighborhood is a simple rectangle compared to the complex lune in metric L_p for $1 < p < \infty$. Now the question is can we make use of this property to design more efficient algorithms? The answer to this question is yes. We present two output sensitive algorithms for computing a β-skeleton in the metrics L_1 and L_∞ for any $\beta \geq 2$. The algorithms are in $O(n^{3/2} \log n + k)$ and $O(n \log n + k)$ respectively, where k is size of the β-skeleton. For the rest of this discussion we shall confine ourselves to the L_∞ metric because rotating by 45^0 and scaling by a factor of $\sqrt{2}$, the L_1 metric is transformed into L_∞.

3.2 Size of a β-skeleton

In the metric L_p for $1 < p < \infty$, a β-skeleton is a subgraph of the Delaunay triangulation [Tou80c] for $\beta \geq 1$. This implies that the size of the β-skeleton is in $O(n)$, for $\beta \geq 1$. But in the L_∞ metric , the size of a β-skeleton is in $\Theta(n^2)$. We show this by constructing an example.

Let us place n points on each vertical edge of a square as shown in Fig. 17. The β-neighborhood of each edge $\overline{pq}$, where $p \in \{p_{i_1}, p_{i_2}, \ldots, p_{i_n}\}$ and $q \in \{p_{j_1}, p_{j_2}, \ldots, p_{j_n}\}$, is empty for any $\beta \geq 1$. Therefore, the size of a β-skeleton for any $\beta \geq 1$ is in $O(n^2)$.

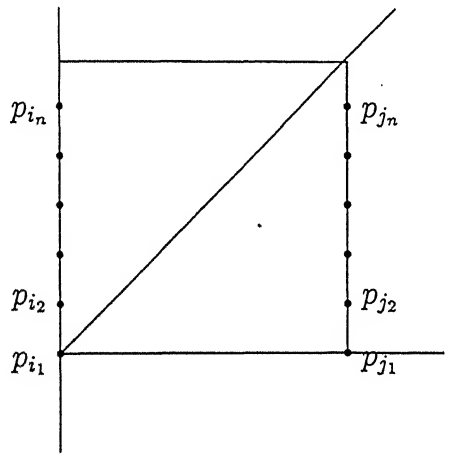


Figure 17: Size of a β-skeleton in the L_∞ metric.

3.3 Algorithm

The β-skeleton can be computed in $O(n^3)$ time by starting with the complete graph as a super graph and pruning the non-empty lunes by a brute force method. This can be improved to $O(n^{5/2} \log n)$ time by the sweepline method described in previous chapter.

In the metrics L_1 and L_∞, the β-lune is a rectangle (see Fig. 18). Therefore, the emptiness query of a lune can be decided efficiently in $O(\log n)$ time using range

tree data structure [PS85]. Hence a β-skeleton can be computed in $O(n^2 \log n)$ time. By taking a closer look into the geometry of the problem, we can improve on this complexity further.

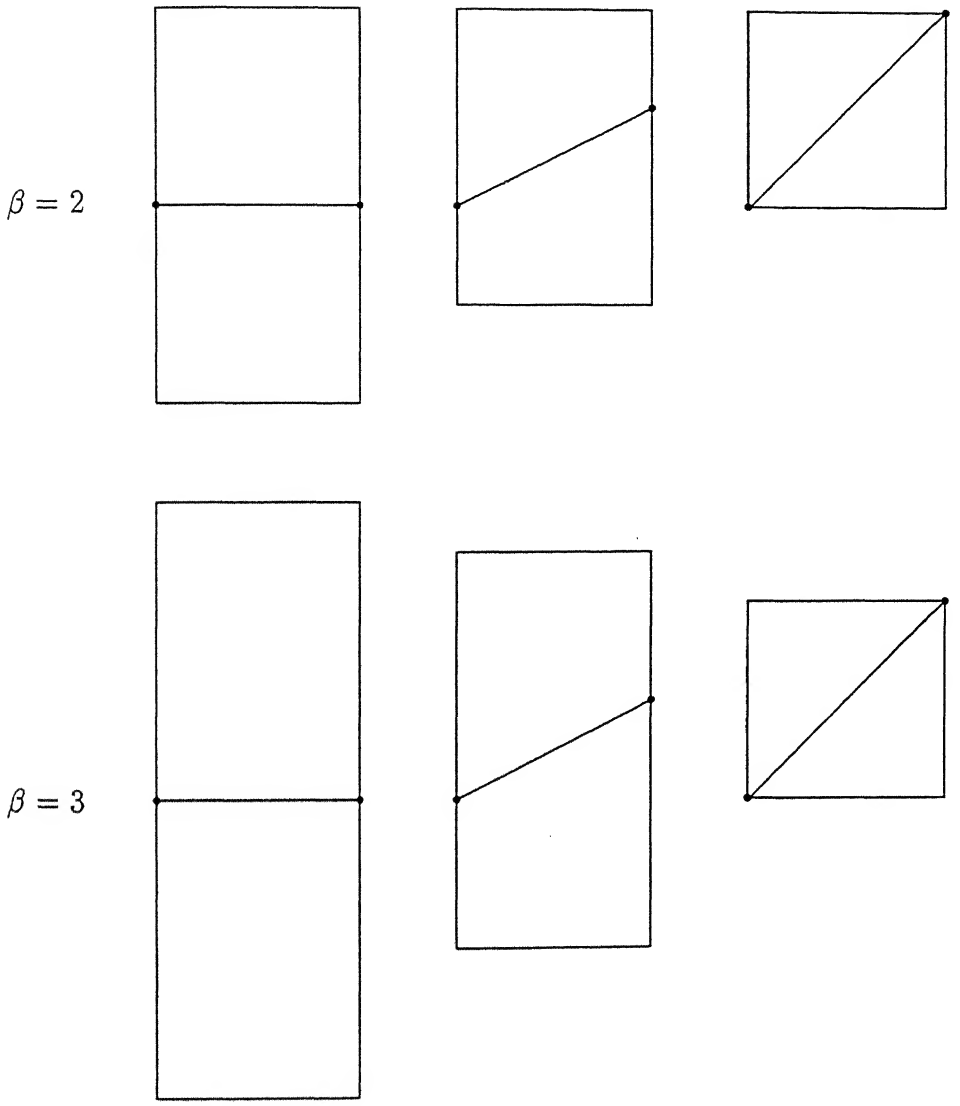


Figure 18: The β-neighborhood in metric L_∞ for $\beta \geq 2$.

The main idea is to find a smaller super graph than the complete graph on P. We adopt Yao's [Yao82] region approach for this.

Consider a point $p_i \in P$. We divide the space around p_i into eight regions [Yao82], by drawing four lines through p_i that make an angle of $0^0, 45^0, 90^0$, and 135^0 respectively with the x-axis. These regions are open on one side and closed on the other. For example, R_1 is closed on the x-axis and open on the diagonal, and R_2 is closed on the diagonal and open on the y-axis. The other regions are similarly defined. These regions are called *narrow regions*, because for any two points p_j, p_k in a region $dist(p_j, p_k) < max\{dist(p_i, p_j), dist(p_i, p_k)\}$. Let the m^{th} region, for $1 \leq m \leq 8$, be labeled $R_m(p_i)$ as shown in the Fig. 19.

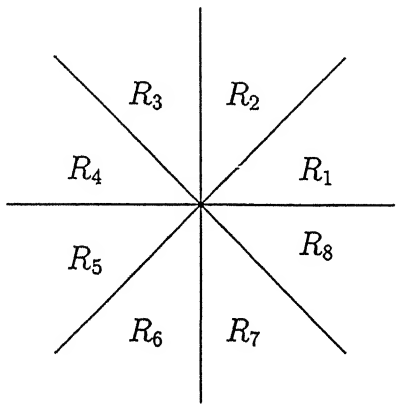


Figure 19: Narrow regions in metric L_∞.

We connect each point p_i to all its nearest neighbors in each region $R_m(p_i)$, for $1 \leq m \leq 8$. The graph so obtained is called the Geographical Neighborhood Graph (GNG). The following lemma is easily verified.

Lemma 3.1 *A β-skeleton for $\beta \geq 2$ is a subgraph of the GNG.*

Proof: From definition we know that any β-skeleton for $\beta \geq 2$ is a subgraph of the β-skeleton for $\beta = 2$. So it is sufficient to show that the β-skeleton for $\beta = 2$ is a subgraph of the GNG.

Let assume that the edge $\overline{p_i p_j} \in \beta$-skeleton for $\beta = 2$. So, $lune(p_i, p_j)$ is empty. This β-lune is defined by two circles of radius $d(p_i, p_j)$, centered at p_i and p_j.

Without loss of generality we assume that $p_j \in R_1(p_i)$. Let $\overline{p_ip_j} \notin GNG$. This implies that there exist a point $p_k \in R_1(p_i)$ such that $d(p_i, p_k) < d(p_i, p_j)$. That is, p_k is inside the circle of radius $d(p_i, p_j)$, centered at p_i.

The definition of the narrow region $d(p_j, p_k) < max\{d(p_i, p_j), d(p_i, p_k)\}$, and $d(p_i, p_k) < d(p_i, p_j)$ implies $d(p_j, p_k) < d(p_i, p_j)$. That is, p_k is inside the circle of radius $d(p_i, p_j)$, centered at p_j.

That is, $p_k \in lune_{\beta=2}(p_i, p_j)$, which is a contribution. Therefore, $\overline{p_ip_j} \in GNG$.

Hence the claim of the lemma. ∎

Now the question is, for a given point how efficiently can we find its nearest neighbors in each region? We consider the region $R_1(p_i)$ for a point $p_i \in P$. The remaining regions $R_2, R_3, \ldots, R_8$ can be handled in the same way.

We sort all the points on their x-coordinate as primary key and y-coordinate as secondary key. The sorted list of points is kept in an array. For each point p_i, we find a nearest neighbor p_j (in $R_1(p_i)$) if it exists in $O(n \log n)$ time using the divide and conquer approach of Guibas and Stolfi [GS83]. The rest of the nearest neighbors can be found as follows.

We compute the intersection points S and N of the boundary of the region $R_1(p_i)$ with the vertical line passing through p_j (see Fig. 20). We locate the points S and N in the sorted array of points. All the points between N and S are the nearest neighbors of p_i in the region $R_1(p_i)$, since points are sorted by y-coordinate as the secondary key. We denote these by $p_{j_1}, p_{j_2}, \ldots, p_{j_k}$.

3.3.1 Pruning edges

Our algorithm for pruning the edges whose lunes are non-empty is based on the following key observation.

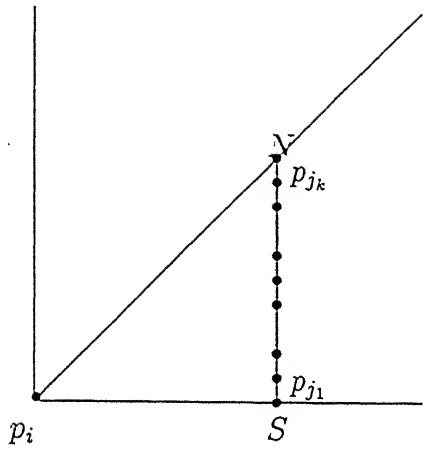


Figure 20: All nearest neighbors of p_i in R_1.

Observation 3.1 *Let $p_{j'}$ be a nearest neighbor of p_i in the region R_1. Then for any $\beta \geq 1$, the $lune_\beta(p_i, p_{j'})$ is a bounded rectangle, contained within an infinite strip, bounded by vertical lines passing through the points p_i and $p_{j'}$.*

Let a_i and b_i be points (of P) inside the strip such that lines through these, orthogonal to the bounding edges of the strip, define the largest rectangle $< r_1, r_2, r_3, r_4 >$ contained in the strip whose interior is empty as shown in the Fig. 21. Note that it's possible that one or both of these points do not exist.

Clearly, all the lunes contained in this rectangle are empty. For $\beta \geq 2$, an easy decision algorithm is based on the following observation.

Observation 3.2 *The regions $lune_\beta(p_i, p_{j_1})$, $lune_\beta(p_i, p_{j_2})$, ..., $lune_\beta(p_i, p_{j_k})$ form a sequence of nested neighborhoods, with $lune_\beta(p_i, p_{j_{l+1}}) \subset lune_\beta(p_i, p_{j_l})$, where $1 \leq l \leq k-1$.*

Here is the algorithm. Starting with the smallest one, $lune_\beta(p_i, p_{j_k})$, we look for the first non-empty lune. If none is found, all the edges between p_i and its nearest neighbors are in the β-skeleton. Else if $lune_\beta(p_i, p_{j_m})$ is the first non-empty lune then the edges joining p_i to $p_{j_m} \ldots p_{j_1}$ are pruned.

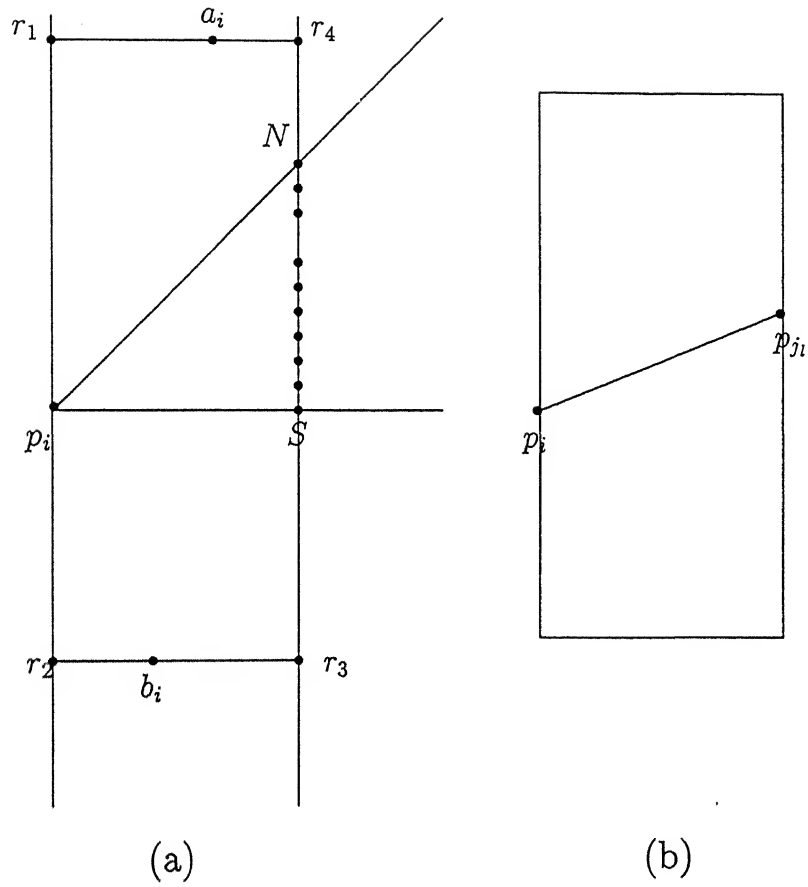


Figure 21: (a) Largest empty rectangle $< r_1, r_2, r_3, r_4 >$ and (b) Empty lune $lune_\beta(p_i, p_{j_l})$ for $\beta = 3$.

Obviously enough, the time required for this procedure is proportional to the number of non-empty lunes.

So all that remains to be done is to show how to compute the points a_i and b_i. Let us do this for the point b_i; the other point a_i can be computed similarly. Since we would like to do this for every point in P, the following abstract problem can be formulated.

> Given a set I of n horizontal intervals and a set P of n points, for each interval $l_i \in I$, find the closest point $b_i \in P$, if any, that lies below it and within the infinite strip determined by the vertical lines that pass through the end points of l_i

Caveat: It just happens that for our problem points and intervals are not distinct; each point is the left end-point of an interval.

We outline two different solutions to this problem: the first based on the familiar sweepline technique, and the second based on the range-tree data structure.

3.3.2 Sweepline Method

We sweep the plane from top to bottom. At any instant, the *sweepline status* $\mathcal{L}$, contains a relative ordering of the end-points of the intervals that have been swept by the sweepline and whose closest point is yet to be found. Two adjacent end-points in $\mathcal{L}$ represent an *elementary interval*. With each elementary interval EI, we associate a list of intervals C_{EI} that contain this elementary interval.

The *event queue* consists of the points of P in y-sorted order.

Each stop in the event queue triggers off two events. The first corresponds to the point p. We locate the elementary interval EI in $\mathcal{L}$ that contains p. The point

p is closest to all the intervals in C_{EI}. We delete these intervals from the sweepline status and adjust the lists of all the elementary intervals.

The second event corresponds to the interval l whose left end-point is p. Let q be its right end-point. The point p divides EI into two intervals EI_1 and EI_2. We delete EI and insert the elementary intervals EI_1 and EI_2 into $\mathcal{L}$. We also create the lists C_{EI_1} and C_{EI_2}. We then do the same for q. For each elementary interval EI' that is completely inside l, we update $C_{EI'}$ by inserting l into it.

We claim the following result.

Lemma 3.2 *The running time of the above algorithm is in $O(n^2 \log n)$; the space complexity in $O(n^2)$.*

Proof: At any time, the sweepline status $\mathcal{L}$ contains at most $2n$ end-points, corresponding to n intervals. We implement $\mathcal{L}$ by a height-balanced tree that stores the end-points at the internal nodes and the elementary intervals at the leaves of the tree.

For each elementary interval $EI \in \mathcal{L}$ we maintain in a height-balanced tree the set of intervals C_{EI} that cover EI.

The event-queue is implemented by an array. It is initialized at the beginning of the sweep in $O(n \log n)$ time. When the sweepline encounters an event, we first process the point p corresponding to this event. We locate the elementary interval $EI \in \mathcal{L}$ that contains p. The time for this is in $O(\log n)$. Since the point p is closest to all the intervals in C_{EI}, these are deleted. If l' is one such interval, we delete its left-end point from $\mathcal{L}$ in $O(\log n)$ time. This causes two adjacent elementary intervals to be merged. The time for this is in $O(\log n)$. The right-end point of l' is similarly deleted. We also delete l' from the interval list of each EI, which is in between the left and right end points of l' in $\mathcal{L}$. This can be done in $O(n \log n)$ time, since there can be at most $O(n)$ elementary interval in sweepline. The cost of

deleting an interval is thus in $O(n \log n)$ time. If we charge this cost to the interval itself, the total cost of deleting all the intervals is in $O(n^2 \log n)$.

We next process the associated interval l. In $O(\log n)$ time we locate the elementary interval $EI \in \mathcal{L}$ that contains its left-end point a and insert it in $\mathcal{L}$ in $O(\log n)$ time. This means splitting EI into two intervals EI_1 and EI_2, deleting EI and inserting EI_1 and EI_2 into $\mathcal{L}$. All this can be done in $O(\log n)$ time.

The interval list C_{EI_1} is the same as C_{EI}, while $C_{EI_2} = C_{EI} \cup \{l\}$. The cost of creating C_{EI_2} is in $O(n)$.

We repeat the above sequence of actions for the right end-point of l.

Further, for each elementary interval EI' that is contained in l, we update $C_{EI'}$ by inserting l into it. This can be done by traversing the sweepline $\mathcal{L}$ from the elementary interval, containing left-end point to the elementary interval containing right-end point. The cost of all these operations is in $O(n \log n)$ time.

Thus the the total cost of processing n intervals is in $O(n^2 \log n)$ time.

Hence this problem can be solved in $O(n^2 \log n)$ time. ∎

The complexity of the above algorithm is prima facie worse than that of the algorithm which, for each given interval, tests by brute-force which of the n points is closest to this interval. The complexity of the latter is in $O(n^2)$. However, if we look at the above algorithm very carefully we notice that the problem lies in considering all the intervals in one go. The cost of maintaining the sweepline status of n intervals is high and dominates the algorithm. The following lemma shows how the complexity can be improved by balancing the processing time for intervals and points.

Lemma 3.3 *Given a set I of n horizontal intervals and a set P of n points, for each interval $l_i \in I$, the closest point $b_i \in P$, which is within the vertical infinite*

strip bounded by the lines passing through the end points of l_i and below l_i can be found in $O(n^{3/2} \log n)$ time using $O(n)$ space.

Proof: We divide the problem into n/m subproblems, each subproblem consisting of m intervals and all n input points. For each of these subproblems the above algorithm takes $O(m^2 \log m + n \log m)$ time. Therefore, the total time required to solve all the n/m subproblems is in $O(nm \log m + n^2/m \log m)$. The minimum time of $O(n^{3/2} \log n)$ is achieved for $m = \sqrt{n}$. The space required is $O(n)$, since we have to consider $\sqrt{n}$ intervals at a time. Hence the claim of the lemma. ∎

3.3.3 Range-tree Method

The complexity can be further improved by using the range-tree data structure [PS85]. For the sake of completeness we briefly describe the method. We preprocess the set of points such that for a given a query interval l_i, the corresponding point b_i, if it exists, can be located quickly. The range-tree is a rooted binary tree which allows us to search along two orthogonal directions, say x and y, simultaneously. It has two levels, the primary and the secondary. The primary level is useful for searching along the x-direction, while the secondary level for searching in the y-direction.

Each non-root node v of the range-tree T corresponds to a set points, lying between the abscissae $L(v)$ and $R(v)$. These points are stored in an array, sorted by their y-coordinates, in the secondary structure $Y(v)$. The range $R(v) - L(v)$ is divided into two sub-ranges such that each sub-range contains at least half the number of points of v. The left and right ranges correspond to the nodes $LSON(v)$ and $RSON(v)$ respectively. The root of T corresponds to whole range, and the secondary level at the root node is a threaded binary tree which contains all the points, sorted by their y-coordinates.

For each non-leaf node v, and for each point p of $Y(v)$, there is a pointer $LBRIDGE(p)$ (respectively $RBRIDGE(p)$) to a point p' of $Y(LSON(v))$ (respectively p'' of $Y(RSON(v))$), which is p itself or or a point that lies immediately below p.

The range-tree can be constructed in $O(n \log n)$ time in a bottom-up manner. The space required is in $O(n \log n)$ since there are $O(\log n)$ levels and each level requires $O(n)$ space.

The application to our problem is quite simple. We narrow down the x-range in which a query interval l_i lies. Each time we do this, we update the point p that is closest to the supporting line of l_i in the current range to the point that is closest in the new range. For this latter work we use the secondary structure stored at each node.

For a given query interval l_i, the details of the search mechanism is as follows. Find the point p which lies immediately below the line passing through l_i by searching in $Y(root)$ in $O(\log n)$ time. For each node v, if l_i is completely within the range of $LSON(v)$ (respectively $RSON(v)$), we search in the sub-tree rooted at $LSON(v)$ (respectively $RSON(v)$). Otherwise, search in both the off-springs. We consider searching in the left off-spring, as the right off-spring can be handled similarly. While proceeding to next level we can get the point which is immediately below the line passing through l_i in $Y(LSON(v))$, by using the pointer $LBRIDGE$. If the range of any node is completely within the given query line segment l, the points which are immediately below l are candidates. Of these, the required point is the one that is closest. See Fig. 22 and Fig. 23. The query time is thus in $O(\log n)$ time.

We summarise our results in the following lemma.

Lemma 3.4 *The algorithm for computing a β-skeleton for $\beta \geq 2$ in region R_1 takes $O(n \log n)$ time.*

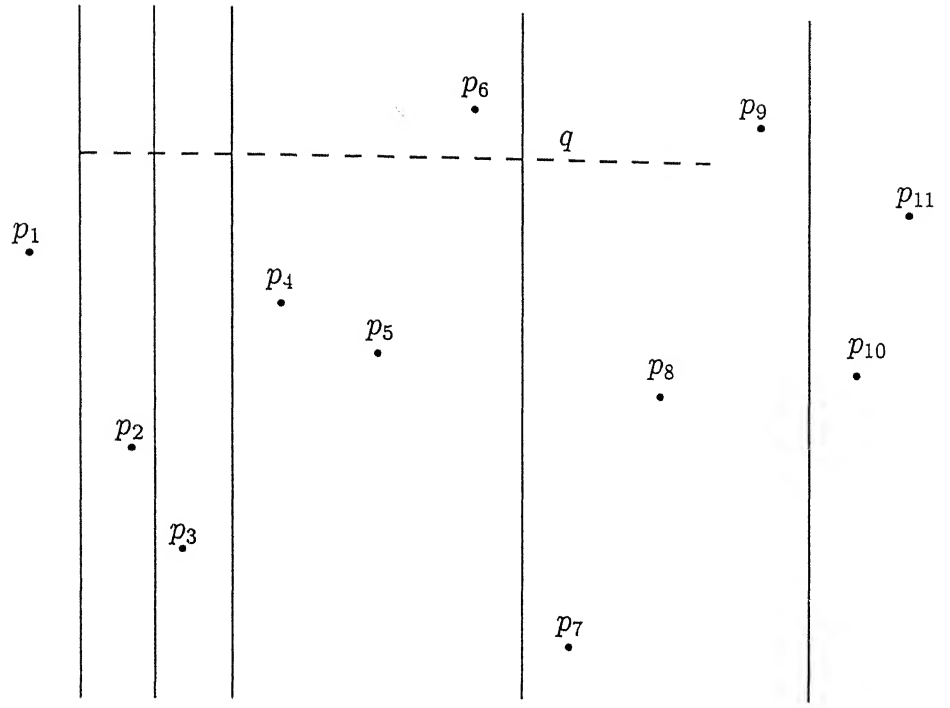


Figure 22: partition into standard intervals.

The above algorithm can be applied to the remaining regions by using an appropriate linear transformation on the input points to get the complete β-skeleton for the given $\beta \geq 2$. Reporting the edges of the β-skeleton takes time proportional to the size of the output. Hence follows the theorem.

Theorem 3.5 *For $\beta \geq 2$, a β-skeleton can be computed in $O(n \log n + k)$ time, where k is size of the output.*

3.4 Conclusion

In this chapter we have described an output sensitive algorithm for computing a β-skeleton in the L_∞ metric for $\beta \geq 2$. The complexity of the algorithm is in $O(n \log n + k)$. It would be interesting to find an output-sensitive algorithm for β in the range $[1, 2)$.

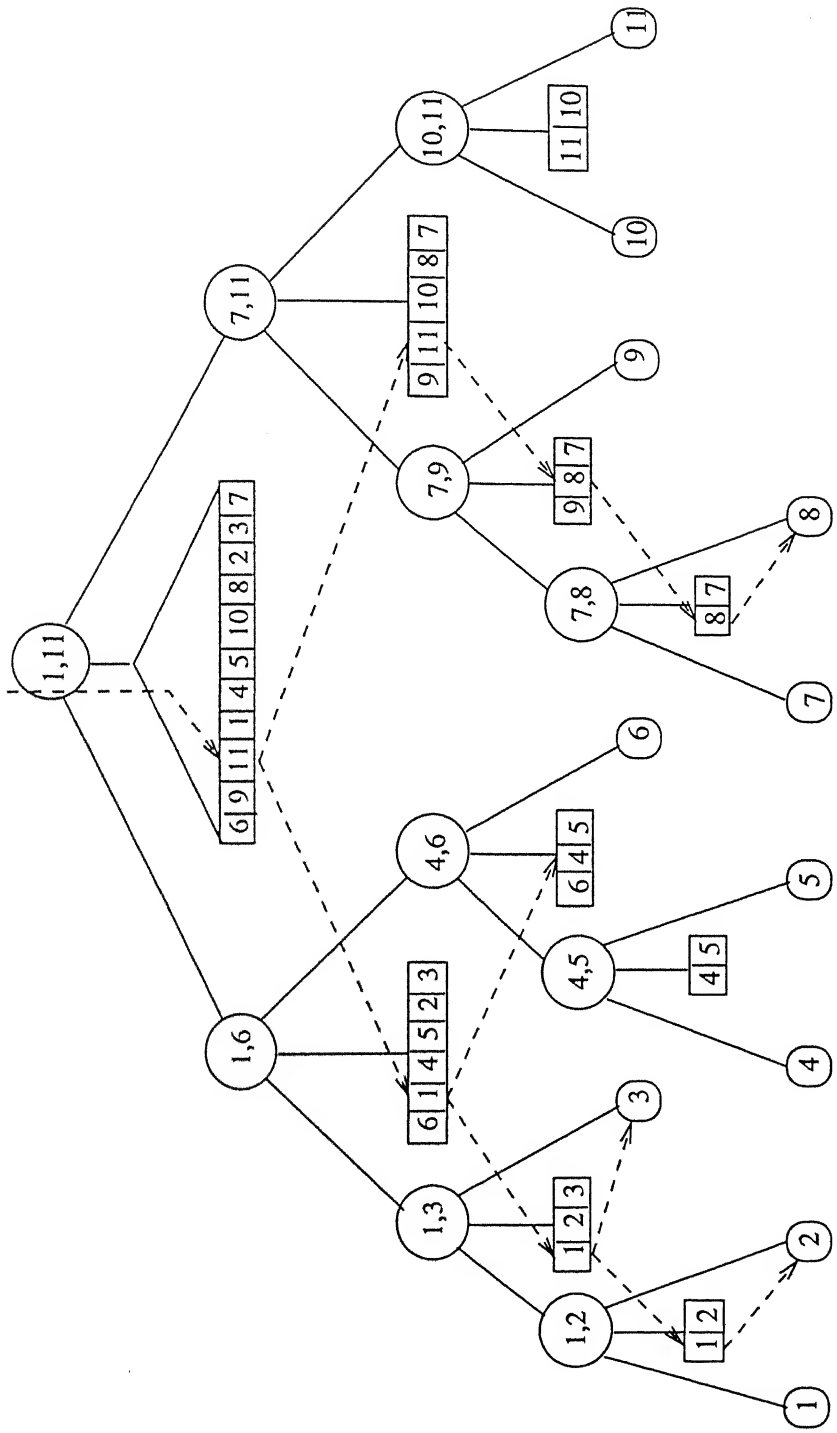


Figure 23: Range tree.

Chapter 4

Computing the β-Spectrum and Their Relatives

4.1 Introduction

In this chapter we propose efficient algorithms for computing the entire β-spectrum of a finite planar point set in the L_2 metric. This means computing for each pair of points the largest β value, β_{max}, for which the β-neighborhood of the pair is empty. The motives are twofold: algorithmic and applied. From an algorithmic point of view this is an interesting problem for which no efficient solution was known before. And from an applied perspective this has rich potential for the problem of character recognition since the spectra of a set of points can be construed as its signature. Moreover, any β-skeleton can be extracted from the β-spectrum in linear time.

Our algorithms are in $O(n^{3/2} \log n + K)$ time for $\beta_{max} \geq 1$ and in $O(n^2 \log n + K)$ time for $\beta_{max} \geq 0$. A point that lies in the β-neighborhood of an edge is a *witness* to this edge. The quantity K is a count of the number of times that the edges of the initial (Delaunay or complete) graph on P are witnessed for $\beta = \infty$.

The rest of the chapter is organized as follows. In the next section we present algorithms based on the techniques of range searching and sweepline. In the two following sections we discuss respectively how to compute efficiently the $k\beta$-spectrum and the additively weighted β-spectrum. We conclude in the fifth and final section.

4.2 β-spectrum

In this section we propose two different algorithms for computing the β-spectrum of P.

The main idea of both the algorithms is to start with a sufficiently large super graph on P and compute the witness set W_e for each edge e, setting $\beta = \infty$. For $\beta \geq 1$ we start with Delaunay triangulation on P since it is a super-graph of the Gabriel graph (corresponds to $\beta = 1$). For $\beta \geq 0$, we start with the complete graph on P. We discuss only the first case as the second case can be handled similarly.

We begin with the key observation that all the witnesses of an edge are contained in the lune that corresponds to $\beta = \infty$. This an open infinite strip bounded by the lines normal to the edge at its end-points. We assume without loss of generality that no edge is horizontal, since all such edges can be handled separately in a preprocessing step in $O(n \log n)$ time.

Let T be a transformation that maps a point p, with coordinates (a, b), into the line T_p with equation $y = -ax + b$ and a line l with equation $y = mx + c$ into the point T_l with coordinates $(-m, c)$ (see Fig. 24). This is known as a pont-line duality transform.

Under this point-line duality transformation T, an infinite strip in the primal plane maps to a vertical line segment in the dual plane. So a point $p \in W_e$ if and only if T_p intersects T_e (see Fig. 25). The problem of computing W_e for an edge e,

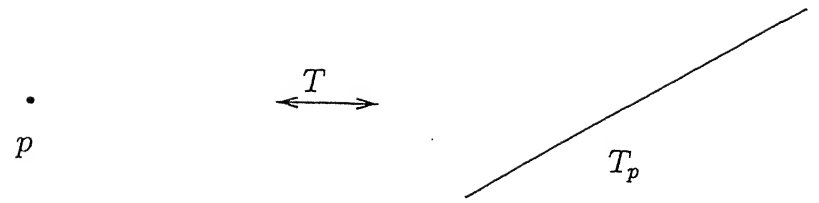


Figure 24: Geometric dual transformation T

therefore, reduces to computing all the lines that intersect a vertical line segment corresponding to e in the dual plane.

We consider two different approaches to the problem.

4.2.1 The Range-Searching Approach

In this approach we attempt to preprocess the set of lines L such that given a vertical query segment q, we can report the lines of L that intersect q in $O(\log n + w_e)$ time, where w_e is the number of intersecting lines.

A naive approach would be to locate one end-point of the query line-segment in the arrangement $\mathcal{A}$ of the lines L, and move to the other end-point, recording all intersections with the the lines of the arrangement. This turns out to be inefficient because the complexity of an individual cell may preclude a constant-time transition to an adjacent cell. But this is precisely what we want, and can be achieved at the cost of some extra preprocessing.

In order to reduce the complexity of an individual cell so that we can move from one cell to an adjacent one in constant time, we first trapezoidize the cells of the arrangement by drawing vertical lines through the vertices of the arrangement. Each such line is terminated by the first line in $\mathcal{A}$ that it meets, if any at all. This can be done using the sweepline technique in $O(n^2 \log n)$ time. The resulting geometric structure is shown in Fig. 26. Let us name this planar subdivision $\mathcal{A}'$.

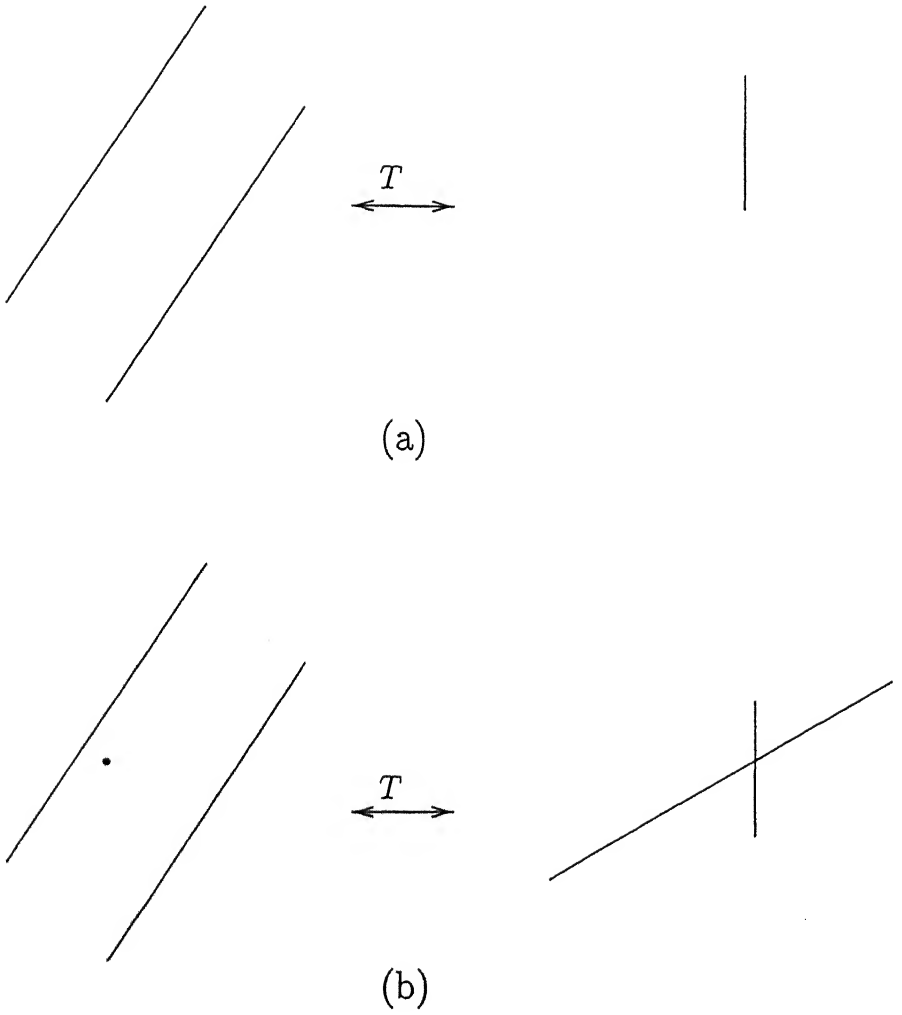


Figure 25: (a) T maps an infinite strip into a vertical line segment, (b) A point $p \in W_e$ if and only if T_p intersects T_e.

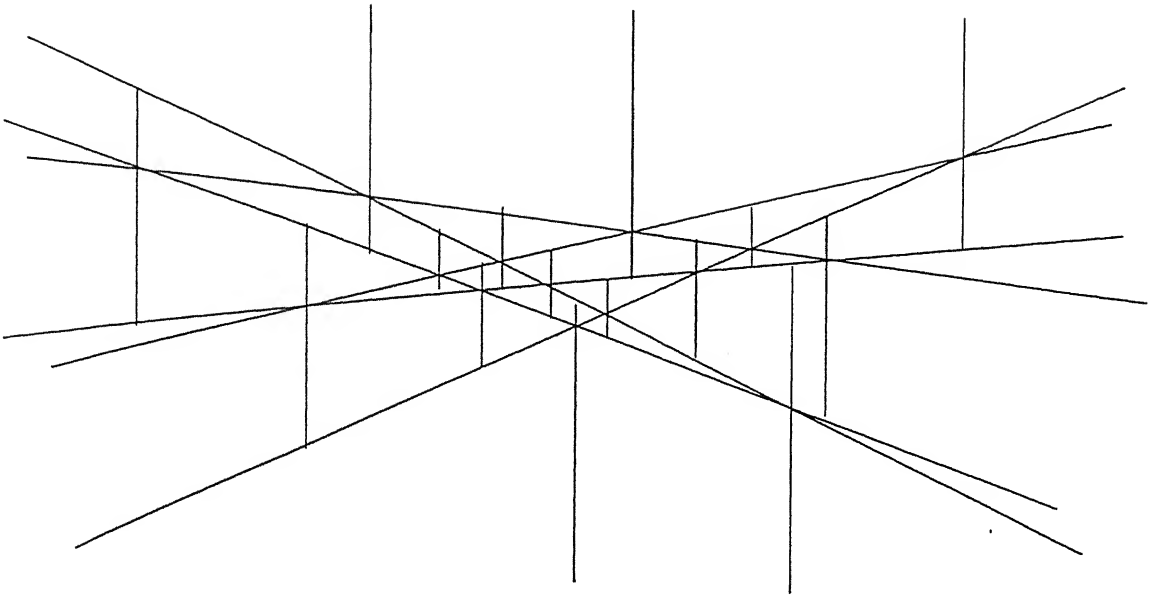


Figure 26: New planar subdivision $\mathcal{A}'$

Though each cell now has the geometric appearance of a trapezium, there can be many vertices on its non-parallel sides (see Fig. 27) in which case it would still be very complex. Such a trapezium has an upper (resp. lower) anomaly if its upper (resp. lower) side consists of more than three vertices.

For an edge $e \in \mathcal{A}$, let $A = \{a_1, a_2, \ldots, a_u\}$ be the set all vertical edges in $\mathcal{A}'$ that are incident on e from above and $B = \{b_1, b_2, \ldots, b_v\}$ be the set of those edges that are incident from below.

For j in $[1 \ldots v-1]$ let $\{a_i, a_{i+1}, \ldots, a_j\}$ be the subset of A whose end-points on e lie strictly between those of b_j and b_{j+1}. We extend the edges $a_{i+1}, a_{i+3}, \ldots, a_{i+l}$, where $l \leq (j-i)$ (see Fig. 28) up to an edge of $\mathcal{A}$ lying immediately below e. While this removes the upper anomalies of all the trapezium that hang on e, it may create upper anomalies for the ones that lie immediately below.

To handle all the regions uniformly, we process $\mathcal{A}'$ by levels of the original arrangement $\mathcal{A}$ from the topmost level to the bottommost one, removing all upper

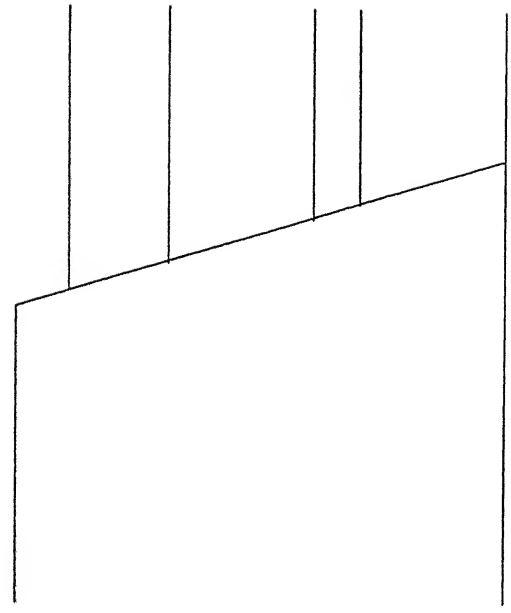


Figure 27: Complex trapezium.

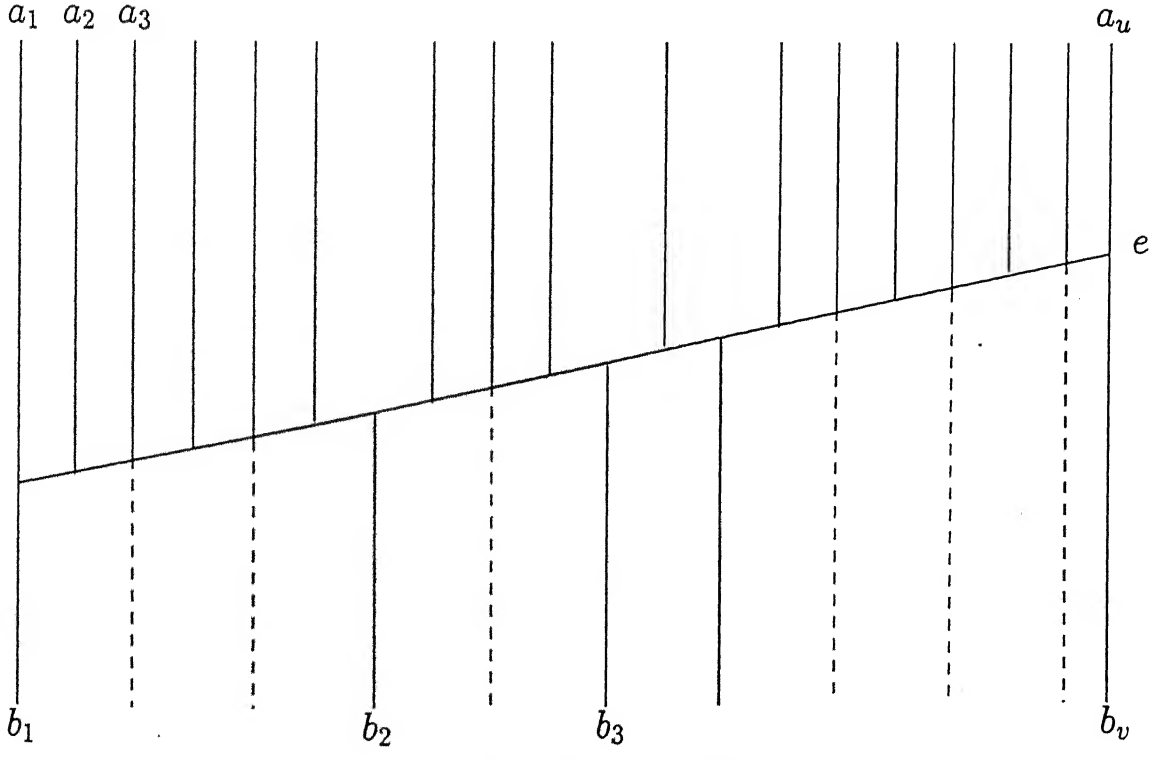


Figure 28: Removing upper anomaly

anomalies. And again from the bottom to the top, removing all lower anomalies. The data structure underlying the resulting planar subdivision $\mathcal{A}''$ is called a hive-graph H [Cha86].

The time and space complexities of this preprocessing step are in $O(n^2 \log n)$ and $O(n^2)$ respectively.

We preprocess $\mathcal{A}''$ for point location in $O(n^2 \log n)$ time [EGS86]. We index the regions of $\mathcal{A}''$, and store these regions in an array R such that the array index of a region is identical to its region index. Along with a region we also store the index of the region just above it. Given a query segment q, we first locate its bottom end-point in $\mathcal{A}''$ in $O(\log n)$ time; we then use its region index to locate it in the array R and determine the index of the region just above. We repeat this till we hit the top end-point of q.

We claim the following result.

Lemma 4.1 *A set of n lines can be preprocessed in $O(n^2 \log n)$ time using $O(n^2)$ space such that given a vertical query segment we can report all lines that intersect it in $O(\log n + w_e)$ query time, where w_e is the size of the output.*

It follows from the above lemma that if we are given a set of n lines L and a set of $O(n)$ vertical line segments V, the intersections of the lines in L with the line segments in V can be computed in $O(n^2 \log n + K)$ amortized time, where K is a count of all such intersections.

The complexity of this algorithm is *prima facie* worse than than that of the straight forward algorithm. But we can whittle away a $\sqrt{n}$ factor from the above complexity by preprocessing groups of lines of size $\sqrt{n}$. The details are in the lemma below.

Lemma 4.2 *Given a set L of n lines and a set V of $O(n)$ vertical segments, computing the intersections of the vertical segments and the lines in L can be done in $O(n^{3/2}\log n+K)$ time using $O(n)$ space, where K is a count of all such intersections.*

Proof: We divide the problem into m subproblems, each consisting of n/m lines and all the vertical segments in V. Solve each subproblem by the algorithm above. Summing up the time complexities of the individual subproblems, we find that the optimal time complexity of $O(n^{3/2}\log\ n + K)$ is achieved for $m = \sqrt{n}$. The space required is $O(n)$ since we are considering $\sqrt{n}$ lines at a time. ∎

Knowing the potential witness set of each edge in DT, we can compute the β_{max} values for each edge e in time proportional to the size of W_e. Therefore β_{max} value for all the edges of DT can be computed in time proportional to K. Thus we have the following result.

Theorem 4.3 *The β-spectrum of a set of points P for $\beta \geq 1$ can be computed in $O(n^{3/2}\log n + K)$ time using $O(n)$ space, where K is a count of the total number of times edges in the DT are witnessed for $\beta = \infty$.*

Similarly, we can obtain the following result.

Theorem 4.4 *The β-spectrum of a set of n points P for $\beta \geq 0$ can be computed in $O(n^2\log\ n)$ time.*

We note that in this case we have to query $O(n^2)$ vertical line segments and hence dividing the lines into groups of size $\sqrt{n}$ does not help.

4.2.2 Sweepline Approach

We can avoid a complicated data structure like the hive-graph, and at the same time take advantage of the fact that we know all query segments beforehand by using a sweepline technique.

In the **sweepline status** $\mathcal{L}$ we store the lines in L according to a total ordering of their intersections. The **event queue** Q is an union of the set of vertical segments V and the intersections of the lines in L, sorted by their x-values.

The algorithm works as follows. As we sweep the structure, if the next event corresponds to an intersection of two lines we update $\mathcal{L}$ by reordering their positions in the intersection-order with the sweepline. If the event is a vertical line segment, we report all the lines in $\mathcal{L}$ that intersect it. This can be done in $O(\log\ n + w_e)$ time, where w_e is the number of lines that intersect the vertical line segment.

As in the range searching technique we also have the following two theorems which follow from the observations above.

Theorem 4.5 *The β-spectrum of a set of points P for $\beta \geq 1$ can be computed in $O(n^{3/2} \log\ n + K)$ time using $O(n)$ space, where K is a count of the total number of times the edges in DT are witnessed for $\beta = \infty$.*

Theorem 4.6 *The β-spectrum of a set of points P for $\beta \geq 0$ can be computed in $O(n^2\ \log\ n + K)$ time using $O(n^2)$ space, where K is a count of the total number of times the edges of complete graph are witnessed for $\beta = \infty$.*

4.3 $k\beta$-Spectrum

As we noted in an earlier chapter, there is a natural generalization of the notion of a β-skeleton to that of a $k\beta$-skeleton obtained by relaxing the emptiness criterion

of a β-lune to let it contain at most $k-1$ points. Similarly, the **$k\beta$-spectrum** is a generalization of the β-spectrum. The $k\beta$-spectrum of a set of n points P is a weighted geometric graph G. Each edge $e \in G$ assigned with largest β value β_{max}, for which the β-neighborhood of the edge contain at most $k-1$ points.

To compute the spectrum for $\beta \geq 1$, we start with the k-Delaunay graph (k-DG) [SC90], known to be a super graph of a $k\beta$-skeleton. To compute the spectrum for β values greater than or equal to 0, we start with the complete graph.

Similar to previous methods we compute the set of witnesses W_e for each edge $e \in k$-DG. For each witness $w \in W_e$ compute the β value at which w becomes witness of the edge e. The β_{max} of the edge e is the kth smallest of these β values, if $|W_e| \geq k$ otherwise $\beta_{max} = \infty$. Therefore, we claim the following result.

Theorem 4.7 *The $k\beta$-spectrum of a set P of n points in the plane for $\beta \geq 1$, can be constructed in $O(kn^{3/2} \log n + K)$ time using $O(n)$ space, where K is a count of the total number of times the edges in k-DT are witnessed for $\beta = \infty$.*

Similarly one can establish the following theorem.

Theorem 4.8 *The $k\beta$-spectrum of a set P of n points in the plane for $\beta \geq 0$, can be constructed in $O(n^2 \log n + K)$ time using $O(n^2)$ space, where K is a count of the total number of times the edges of the complete graph are witnessed for $\beta = \infty$.*

4.4 Weighted β-Spectrum

Let w_i be the additive weight of the point $p_i \in P$. Each weighted point p_i can be treated as a circle B_i, centered at p_i, with radius equal to its weight w_i. The weighted distance $d_w(p_i, x)$, between a point $p_i \in P$ and an arbitrary point x, is

$d(p_i, x) - w_i$ if x is outside B_i, and zero otherwise. The weighted distance $d_w(p_i, p_j)$, between any two points $p_i, p_j \in P$ is the distance between B_i and B_j. Let D denote the set of circles B_i.

The additively weighted β-spectrum is a weighted geometric graph, each edge e being weighted with the largest β value, β_{max}, for which the β-neighborhood of e is not intersected by any weighted circle.

In the rest of this section we discuss an algorithm for computing the weighted β-spectrum.

4.4.1 An Algorithm

To compute the spectrum for $\beta \geq 1$ we start with a super graph which is the weighted Delaunay graph (WDG) [Mir93]. For $\beta \geq 0$, we start with the complete graph. We discuss only the first case.

For each edge $e \in$ WDG, we compute the set of witnesses for $\beta = \infty$. The previous algorithms do not generalize in a straight forward manner, because a circle in the primal plane does not have a dual that is a simple object. However, a simple observation helps us get around this problem. To explain this, we need the help of some notations. The symbol $S_{\overline{p_i p_j}}$ denotes the $lune_\infty(p_i, p_j)$; we will let S_k^e denote the smallest infinite strip which encloses the weighted circle B_k and is parallel to S_e.

The following observation is very straightforward.

Observation 4.1 *A point p_k is witness to an edge e for some value of β if and only if S_k^e intersects S_e.*

The standard point-line dual transformation T maps a non-vertical infinite strip into a vertical line segment. Moreover, two parallel non-vertical infinite strips

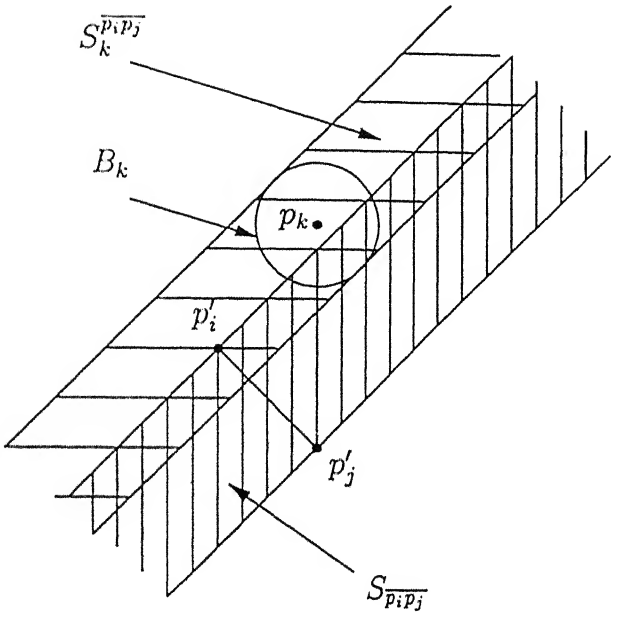


Figure 29: Illustration of Observation 1.

intersect each other in the primal plane if and only if their corresponding vertical line segments overlap in dual plane. In other words, a point p_k is witness of an edge e for some value of β if and only if the vertical line segments $T_{S_k^e}$ and T_{S_e} overlap each other in the dual plane. We use the simpler notations I_k^e and I_e for $T_{S_k^e}$ and T_{S_e} respectively. In terms of this simpler notation the above observation may be restated as below:

Observation 4.2 *A point p_k is witness to an edge e for some value of β if and only if I_k^e intersects I_e.*

Therefore, to compute the set of witnesses of an edge $e \in$ WDG, we find all the vertical line segments I_k^e that intersect the vertical segment I_e in the dual plane. The question now is how efficiently can we get all the vertical segments I_k^e's for each $e \in$ WDG ? The following observations provide an insight. Their proofs are immediate from the definition of the dual transform.

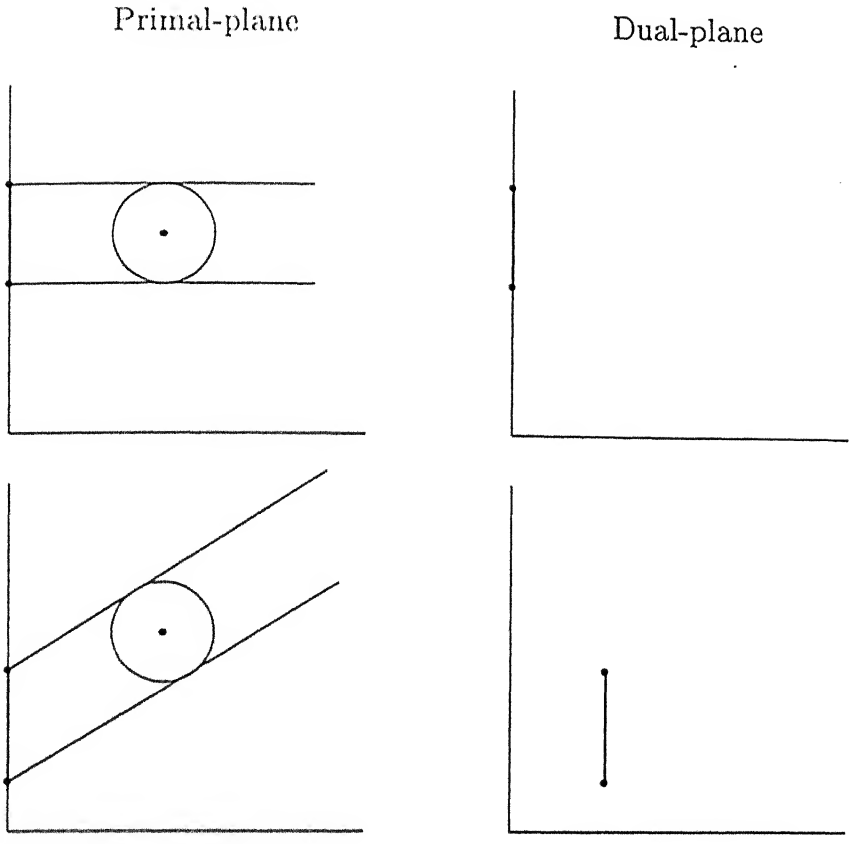


Figure 30: Rotation of a strip I_i and its effect in dual plane.

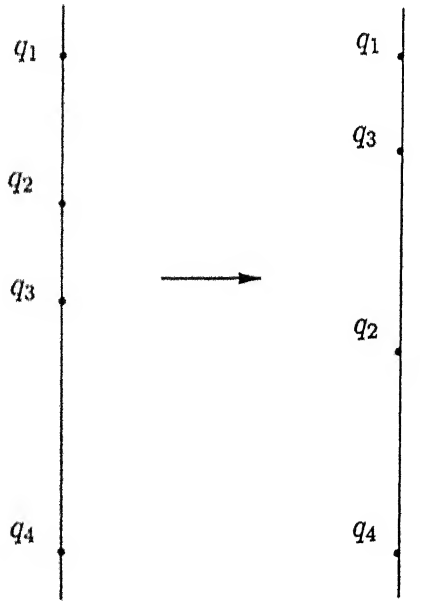


Figure 31: Event updation.

as follows: if q_2 is the top end point of I_i^x, delete I_i^x from C_{EI_2}, since I_i^x is below EI_2. Otherwise it is the bottom end point of I_i^x; insert I_i^x in C_{EI_2}, since EI_2 is in I_i^x. Similarly update C_{EI_2} with respective to the point q_3. There is no change in the interval lists, C_{EI_1} and C_{EI_3}.

When the sweepline $\mathcal{L}$ coincides with I_e, we report all the intersecting intervals. This is done as follows. We find the elementary interval $EI \in \mathcal{L}$ such that the top end point of I_e is in EI. The interval I_e intersects all the intervals in C_{EI}. This takes care of all intervals, whose top end point is above EI. But the top end-points of some intersecting intervals may start below that of EI. These can be reported by going down from the top end-point to the bottom end-point of I_e in the sweepline status $\mathcal{L}$. During this, if the top end-point of any interval is encountered we report the interval corresponding to this top end-point.

It is easy to determine the events arising out of the adjacency of a pair of points on the sweepline status $\mathcal{L}$. This is the x-value corresponding to the slope of a common tangent to the weighted circles (in the primal plane) of the intervals (in the dual plane) to which these end-points belong. This can be done in $O(1)$ time for each event (see Fig. 32).

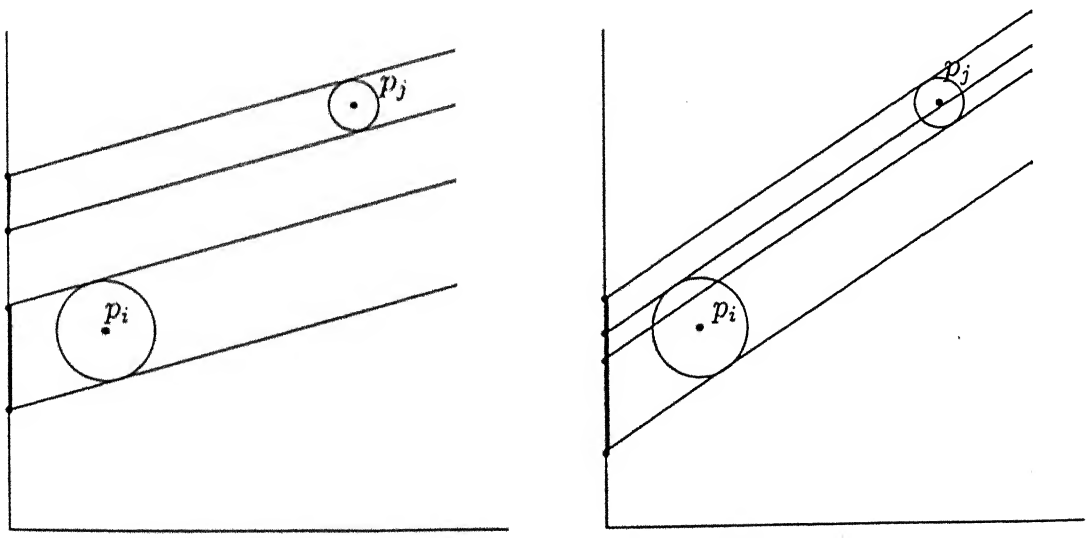


Figure 32: Finding an event.

Lemma 4.9 *The witness set for all the edges $e \in WDG$ can be computed in $O(n^2 \log n + K)$ time using $O(n^2)$ space, for $\beta = \infty$.*

Proof: At any time, the sweepline status $\mathcal{L}$ contains $2n$ end points of n intervals. $\mathcal{L}$ can be implemented simply by an array of size $2n$. At $x = 0$, the sweepline status $\mathcal{L}$ can be initialized in $O(n \log n)$ time. Each pair of adjacent end points in $\mathcal{L}$ forms an elementary interval. For each elementary interval $EI \in \mathcal{L}$ we compute a set of intervals C_{EI}, and maintain them in a height-balanced tree. All this can be done in $O(n^2 \log n)$ time.

The initial events are determined by adjacent end-points in $\mathcal{L}$. We compute these events by traversing the sweepline $\mathcal{L}$, from one end to other in $O(n)$ time. We also add all the vertical segment I_e for all $e \in$WDG. This initializes the event-queue; the complexity of this is in $O(n \log n)$.

Let e be an event which is a change of order of q_2 and q_3 (see Fig. 31). The cost of locating these point in $\mathcal{L}$, and hence the three consecutive elementary intervals EI_1, EI_2, and EI_3 such that $[q_2, q_3] = EI_2$ is in $O(\log n)$. Updating C_{EI_2} can be done in $O(\log n)$ time, since we need to make at most two deletions and two insertions to the list C_{EI_2}. Inserting the next events of q_1 and q_3, q_3 and q_2, and q_2 and q_4 into Q can be done in $O(\log n)$ time. There are at most $O(n^2)$ events since an interval can intersect another interval at most constant number of times. Therefore the total cost of processing all such events is in $O(n^2 \log n)$ time.

When the sweepline $\mathcal{L}$ coincides with I_e, in $O(\log n)$ time we locate the elementary interval $EI \in \mathcal{L}$ such that the top end-point of I_e is in EI. We report all intervals in C_{EI} and travel down the sweepline $\mathcal{L}$ from the top end-point of I_e to the bottom end-point of I_e, reporting intervals whose top end-point are encountered. This can be done in $O(w_e)$ time since there are at most $2w_e$ end points between the top and bottom end-points of I_e, where w_e number of intervals I_e intersected. The total cost of processing all the vertical segments I_e for $e \in$WDG, is in $O(n \log n + K)$

time, where K is a count of the number of times that the edges of the WDG are witnessed for $\beta = \infty$. ∎

The complexity of the above algorithm is prima facie worse than the brute-force algorithm which tests a circle for intersection with each of the $O(n)$ infinite strips, corresponding to each edge of the WDG! For the complexity of this is in $O(n)$, and therefore in $O(n^2)$ for n circles. However, if we look at the above algorithm very carefully we notice that the problem lies in considering all the intervals in one go. The cost of maintaining the sweepline status of n intervals is high and dominates the algorithm. So, similar to previous algorithms here also we balance the processing time for intervals and finding the witnesses to improve the time complexity. One can easily verify the following lemma.

Lemma 4.10 *The set of witnesses for each edge $e \in WDG$ can be computed in $O(n^{3/2} \log n + K)$ time using $O(n)$ space, where K is is a count of the total number of times edges in WDG are witnessed for $\beta = \infty$.*

This gives us the following theorem.

Theorem 4.11 *For $\beta \geq 1$, the weighted β-spectrum of P can be constructed in $O(n^{3/2} \log n + K)$ time using $O(n)$ space, where K is is a count of the total number of times edges in WDG are witnessed for $\beta = \infty$.*

Similarly, the weighted β-spectrum of P, for $\beta \geq 0$, can be constructed by starting with the complete graph on P as the super graph and find set of witnesses for each edge. The following theorem is easily proved.

Theorem 4.12 *The weighted β-spectrum of a set P of n points in the plane, for $\beta \geq 0$, can be constructed in $O(n^2 \log n + K)$ time using $O(n^2)$ space, where K is is a count of the total number of times edges are witnessed for $\beta = \infty$.*

4.5 Conclusion

In this chapter we have proposed efficient algorithms for computing the β-spectrum, $k\beta$-spectrum and, weighted β-spectrum in the L_2 metric. It would be interesting to implement these algorithms and test them on some practical data.

Chapter 5

β-Skeletons in Higher Dimensions

5.1 Introduction

In this chapter we extend the definition of a β-skeleton to higher-dimensions. We show that the size of a lune-based β-skeleton for $\beta > 2$ in higher dimensions is linear. We also present an efficient algorithm for computing a β-skeleton for $\beta > 2$.

The rest of the chapter is organized as follows. In the next section we define the two types of β-skeletons - lune and sphere-based. In the following section, we prove an upper bound on the size of a β-skeleton for $\beta > 2$. We next present an efficient algorithm for computing a β-skeleton for $\beta > 2$. We conclude in the third and final section.

5.2 Definitions

A β-skeleton is a geometric graph, obtained by joining pairs of points whose β-neighborhoods are empty. The neighborhood size depends on the parameter β, and

are of two types.

The sphere-based β-neighborhood of a pair of points (p_i, p_j) is defined as follows: for $\beta \geq 1$, it is the union of all open spheres of radius $\beta|\overline{p_i p_j}|/2$, passing through p_i and p_j. For $\beta \in [0, 1]$, it is the **interior** of the intersection of all open spheres of radius $|\overline{p_i p_j}|/(2\beta)$, passing through p_i and p_j.

The lune-based β-neighborhood of a pair (p_i, p_j) of points is defined as follows: for $\beta \geq 1$, it is the **interior** of the intersection of two spheres of radius $\beta|\overline{p_i p_j}|/2$, centered at the points $(1 - \beta/2)p_i + (\beta/2)p_j$ and $(\beta/2)p_i + (1 - \beta/2)p_j$, respectively. For $\beta \in [0, 1]$, the lune-based neighborhood is identical with the sphere-based neighborhood.

5.3 Algorithm

5.3.1 Main Ideas

The essential technique is to compute a small super graph of a β-skeleton and then prune the edges of this super graph that do not belong to the β-skeleton.

To accomplish the first step we use the powerful region approach of Yao [Yao82]. This gives us a super graph, called the Geographical Neighborhood Graph (GNG, for short).

In the section below, we discuss an improved version of Yao's approach. To aid the intuition we discuss first the two-dimensional case and then indicate how the result can be extended to higher dimensions.

5.3.2 The two-dimensional case

The key idea of this approach is to decompose the neighborhood of a point p_i into a finite number of sectors by drawing rays from p_i. Each sector is said to be a narrow region and is characterized by the fact that if p_j and p_k are two points in a sector then $dist(p_j, p_k) < max(dist(p_i, p_j), dist(p_i, p_k))$. This latter inequality can be guaranteed by letting the angle of each sector be $\leq 60^0$. The nice consequence is that in each sector only the edges connecting p_i to its nearest neighbors are edges of the super graph.

Our observation is that the upper bound of 60^0 degrees is a function of the parameter value β. And though the inequality characterizing a narrow region need not hold when we raise the upper bound, the following is true.

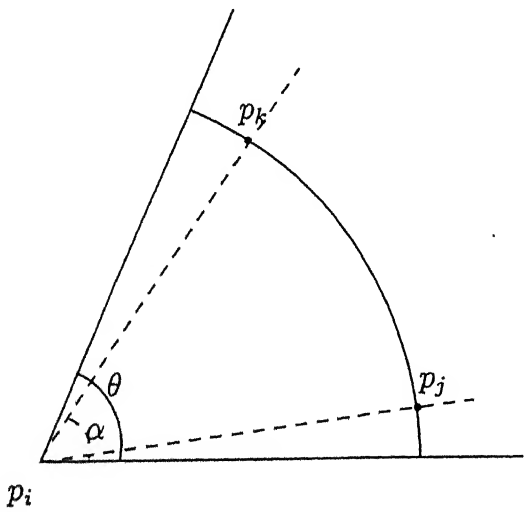


Figure 33: Region approach

Lemma 5.1 *For a given point p_i, let the angle of a sector R be $\theta = \arccos(1/\sqrt{2\beta})$. If more than one point in R is nearest to p_i, then none of the edges, connecting p_i to its nearest neighbors is in a β-skeleton for $\beta > 2$.*

Proof: Let the points p_j and p_k be two nearest neighbors of the point p_i in the region R (see Fig. 33). Let the angle $\angle p_j p_i p_k = \alpha$. The points C_i and C_j are the

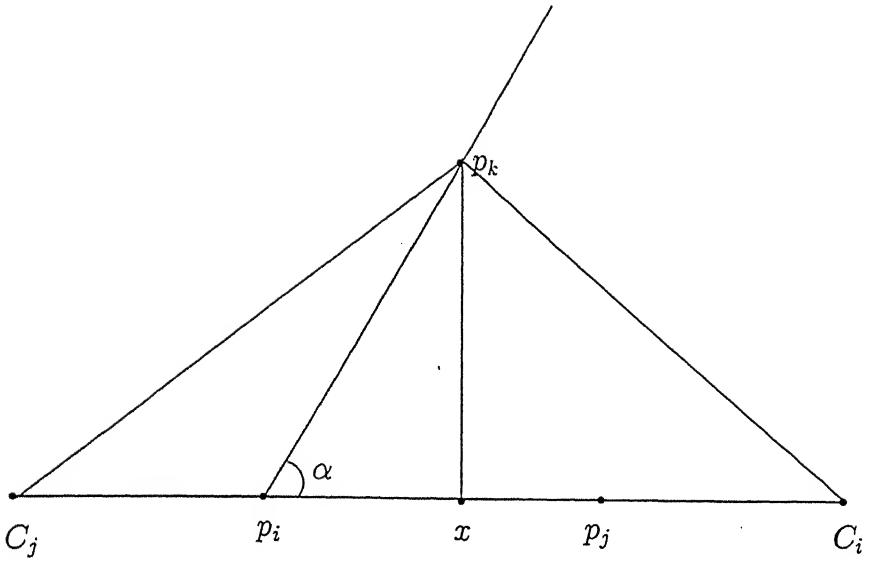


Figure 34: Region approach

centers of circles whose radius is $\beta|\overline{p_ip_j}|/2$ and pass through p_i and p_j respectively, as shown in Fig. 34. Elementary geometry shows that the point p_k is inside the β-neighborhood of p_i and p_j; for it is trivial that C_jp_j is longer than C_jp_k. And the fact that $p_j \in \beta$-neighborhood(p_i, p_k), follows from the upper bound on the angle α.

∎

Therefore, we divide the region around a point p_i by rays that emanate from p_i, making angles of $(m-1)\arccos(1/\sqrt{2\beta})$ with the x-axis, where $1 \leq m \leq 6$ if $\theta < 72$ and $1 \leq m \leq 5$ otherwise. We number the regions between two successive rays $(m-1)\arccos(1/\sqrt{2\beta})$ and $m\arccos(1/\sqrt{2\beta})$ by m and label the region bounded by these as $R_m(p_i)$.

An important consequence of the above lemma is that we get a super graph of linear size for $\beta > 2$.

To construct the GNG, for each point p, we have to find its nearest neighbors in each sector R_i. We show how to do this for the region R_1 (the remaining regions can be handled similarly). We rotate the y-axis by an angle $90^0 - \theta$. This is equivalent

to transforming each point p into a new point p' by the following transformation.

$$\begin{bmatrix} p_{x'} \\ p_{y'} \end{bmatrix} = \begin{bmatrix} 1 & -\cot(\theta) \\ 0 & 1/\sin(\theta) \end{bmatrix} \begin{bmatrix} p_x \\ p_y \end{bmatrix}$$

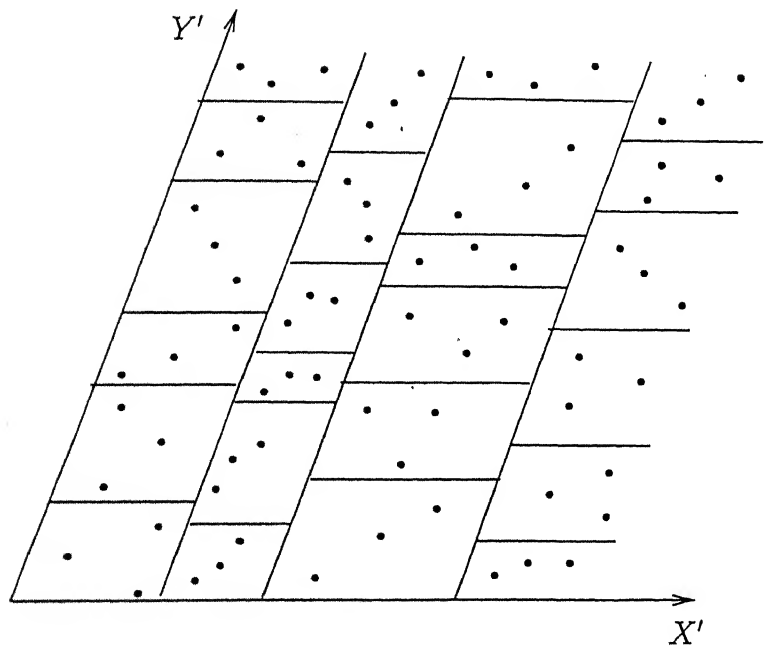


Figure 35: Divide the points into cells.

We sort the new points by their X-coordinates, and divide them into m consecutive subsets of equal size. We sort the points in each subset by their new Y-coordinates and divide each subset into m consecutive subsets of equal size as shown in Fig. 35. For each cell c_i, we compute the following four attributes:

$$X_{max}(c_i) = max\{p_{x'} | p \in c_i\}$$

$$Y_{max}(c_i) = max\{p_{y'} | p \in c_i\}$$

$$X_{min}(c_i) = min\{p_{x'} | p \in c_i\}$$

$$X_{min}(c_i) = min\{p_{y'} | p \in c_i\}$$

With respect to p, the m^2 cells are divided into three classes. A cell is in class 1, if all its points are in $R_1(p)$. If all its points are outside $R_1(p)$, it is in class 2;

otherwise in class 3. To compute the nearest neighbors of p in $R_1(p)$ we use following procedure: cells of class 2 can be ignored. We compute the Voronoi diagram for each cell of class 1 in $O(n/m^2 \log(n/m^2)$ time per cell, since the number of points in each cell is in $O(n/m^2)$. We locate p in each cell's voronoi diagram and find the nearest neighbor in each cell in $O(\log(n/m^2))$ time. We scan all the points in cells of class 3 and find the nearest neighbor of p that is in $R_1(p)$.

Therefore, the total time required for computing nearest neighborhood in R_1 for each point is in $O(n \log(n/m^2) + n m^2 \log(n/m^2) + n^2/m)$. Choosing $m = (n/\log n)^{1/3}$, the time complexity is seen to be in $O(n^{5/3} \log^{1/3} n)$.

5.3.3 The higher-dimensional case

To extend the above discussion to higher dimensions, we need the notion of a basis and a frame as introduced by Yao [Yao82]. We consider our metric space to be a d-dimensional real vector space. If $B = b_1, b_2, ..., b_d$ is a basis of this vector space, then the convex cone, $Conv(B) = \{\sum_{i=1}^{d} \lambda_i b_i | \lambda_i \geq 0 \text{ for all } i\}$ corresponds to what we called a sector in the two-dimensional case. A set of bases $B_1, B_2, ..., B_k$ form a frame, $\mathcal{F}$, if $\bigcup Cone(B_i)$ is the entire vector space.

The angle of a cone is the maximum angle between any two vectors in this cone and the angle of the frame is the maximum angle taken over all the cones.

Yao proved the interesting result that if ψ is any angle in the range $0 < \psi < \pi$, we can always construct a frame whose angle is less than ψ. For our problem, we need to construct a frame whose angle is less than $\arccos(1/\sqrt{2\beta})$. The independence of this angle with respect to dimension is a bit of a surprise.

As in the 2-dimensional case, the following lemma identifies the possible edges of a β-skeleton for $\beta > 2$.

Lemma 5.2 *For a given point p_i, let the angle of a cone R be bounded by $\theta = \arccos(1/\sqrt{2\beta})$. If more than one point in R is nearest to p_i, then none of the edges, connecting p_i to its nearest neighbors is in a β-skeleton for $\beta > 2$.*

Proof: Let the points p_j and p_k be two nearest neighbors of the point p_i in the region R. Let h be a plane passing through the points p_i, p_j, and p_k. The intersection of h and the β-neighborhood(p_i, p_j) is a 2-dimensional β-neighborhood. Therefore, the point p_k is in β-neighborhood(p_i, p_j) from Lemma 5.1. Similarly, we can show that the point p_j is in the β-neighborhood(p_i, p_k). ∎

As in the 2-dimensional case, we conclude that for fixed d, the size of the β-skeleton is in $O(n)$ for $\beta > 2$.

Given a frame $\mathcal{F}$, the GNG can be computed using the method proposed by Yao in [Yao82]. Here is a brief sketch of his algorithm. Cnsider a basis B_i. We transform the coordinates of our points to this new basis B_i. Sort the points on the first coordinate and divide them into m consecutive subsets of equal size. In each subset, sort the points on the second coordinate and divide these into m consecutive subsets of equal size. Repeat this processes for the remaining $d-2$ coordinates. This divides the points into m^d subset of equal size in $O(dn \log n)$ time. We preprocess each subset such that, given a query point p, we can find a nearest neighbor in $O(\log(n/m^d))$ time. For each point p, its nearest geographical neighbors can be computed as follows. Imagine the frame $\mathcal{F}$ to be placed at p. Consider a cone R. With respect to R, the m^d cells are divided into three classes. A cell is in class 1, if all its points are in R. If all its points are outside R, it is in class 2; otherwise it is in class 3. To compute the nearest neighbors of p in R we use following procedure: cells of class 2 can be ignored. We locate p in the Voronoi diagram of each cell in class 1 and find the nearest neighbor in $O(\log(n/m^d))$ time for this cell. We scan all the points in the cells of class 3 and find a nearest neighbor of p that is in R. This process, repeated for all the cones at p and all the input points, gives us the GNG. Following the analysis of [Yao82], we can prove the following result.

Lemma 5.3 *Given a set P of n points, a super graph of the β-skeleton for $\beta > 2$ can be computed in $O(n^{2-a(d)}(\log n)^{1-a(d)})$ time, where $a(d) = 2^{-(d+1)}$.*

5.3.4 Pruning the GNG

A brute-force method of thinning the super graph to the β-skeleton is to check the status of each edge. This means checking if the lune corresponding to an edge is empty or not. Since this requires $O(n)$ time and there are $O(n)$ edges, the complexity is in $O(n^2)$.

To improve on this result, we consider all the spheres that determine the lunes of the edges in GNG. The resulting arrangement of spheres divides the space into $O(n^d)$ cells. We can preprocess these hyperspheres, to obtain for each k-face, $0 \leq k \leq d$, the set of lunes that contain this k-face. For each point p, locate the face f in the arrangement of hyperspheres and prune the all the lunes that contain f. That is, the problem of checking emptiness of lunes is reduced to point location in an arrangement of hyperspheres.

Since an arrangement of hyperplanes is easier to work with, by an inversive transform we map the spheres to a set of hyperplanes in $d + 1$ dimensions. We briefly discuss the inversion transformation.

An inversion is determined with respective to two parameters. These are the center and radius of inversion. Consider the planar case, with the center of inversion at the origin and radius of inversion $l > 0$. A point p is mapped into another point p' such that $d(o,p)\, d(o,p') = l$. In other words, a vector in the direction θ is mapped to another vector in the same direction but with inverted magnitude. Similarly, a circle which passes through the center of inversion is mapped to a line which does not pass through the center of inversion. Moreover, the interior of the circle is mapped to a half-plane as shown in the Fig. 36.

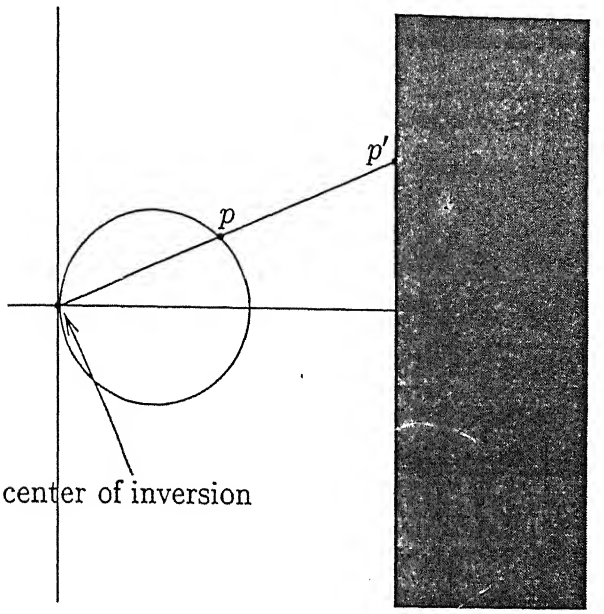


Figure 36: Inverse transformation in plane.

The inversion of our hyperspheres to hyperplanes in a space that is one dimensional higher is carried out this way. Imagine our spheres to be embedded in a d-dimensional hyperplane in $d+1$-space. Let c be a center of inversion, lying outside this hyperplane. Consider a hypersphere, S', that passes through this center of inversion and a given sphere, S, that lies embedded in the d-dimensional hyperplane. The d-dimensional hyperplane, that S' inverts to with repect to c corresponds to S.

Now the problem of point location in the arrangement of hyperspheres in d-space is reduced to the problem of point location in the arrangement of hyperplanes in $(d+1)$-space. Using the generalized binary search method for higher dimensions, proposed by Dobkin and Lipton [DL76], we can preprocess these hyperplanes for point location query. We briefly describe the method, first in 2-dimensions and then in higher dimensions.

Given a set of n lines in the plane, these lines are preprocesed for point location. The projections of the intersections of these lines on the x-axis are sorted. These projected points divide the x-axis into $O(n^2)$ intervals, such that in each interval no two given lines intersect. Therefore, for each interval store the relative ordering of

the given lines. The time required is in $O(n^2 \log n + n^2 n \log n)$. The query can be answered in $O(\log n + \log n^2)$ time. That is, $O(3 \log n)$ time.

We proceed similarly in $(d+1)$-space. Project the intersections of all pairs of the given n hyperplanes on to a hyperplane, which is distinct from given hyperplanes. In each region in the arrangement of these $O(n^2)$ projected hyperplane, no two original hyperplanes intersect. So for each region store the relative ordering of these hyperplanes. To locate the region in $O(n^2)$ hyperplanes in d-dimension, we recursively solve the problem.

Let $\mathcal{P}(n, d+1)$ be the time required for preprocessing the n hyperplanes in $(d+1)$-dimension. We get the following recurrence relation for $\mathcal{P}(n, d)$:

$$\mathcal{P}(n, d+1) = \mathcal{P}(n^2, d) + O(n^{2d} n \log n)$$

with $\mathcal{P}(n, 2) = O(n^2 n \log n)$.

Let $\mathcal{Q}(n, d+1)$ the time required for answering a query. We have the following recurrence relation for $\mathcal{Q}(n, d)$:

$$\mathcal{Q}(n, d+1) = \mathcal{Q}(n^2, d) + O(\log n)$$

with $\mathcal{Q}(n, 2) = O(\log n)$.

The solutions to these recurrences allow us to conclude the following.

Lemma 5.4 *A set of n hyperplanes in $(d+1)$-dimension can preprocessed in $O(n^{2^{d+1}-1}$ l time so that each subsequent point location query can be answered in $O((2^{(d+1)-2} + (d-1)) \log n)$ time.*

The complexity of the above algorithm is worse than the brute force algorithm for pruning the lunes. However, if we look at the above algorithm carefully we notice that the preprocessing time for $O(n)$ hyperplanes is high and dominates the

algorithm, whereas the query time is low. We can improve on the complexity by balancing the preprocessing and query times to derive the following result.

Lemma 5.5 *Given a GNG, the nonempty lunes can be eliminated in* $O(n^{2-a'(d)} \log n)$ *time for* $\beta > 2$, *where* $a'(d) = (2^{d+1} - 1)^{-1})$.

Proof: We divide the problem into $O(n/m)$ subproblems, each subproblem consisting $O(m)$ hyperplanes and all the input points. For each of these subproblems, we preprocess the hyperplanes for point location in $O((n/m)\, m^{1/a'(d)} \log m)$ time. The total point location time is in $O(n^2/m \log m)$. Therefore the total time required to solve all the $O(n/m)$ subproblems is in $O((n/m)\, m^{1/a'(d)} \log m + n^2/m \log m)$. The minimum time of $O(n^{2-a'(d)} \log n)$ is achieved for $m = n^{a'(d)}$. ∎

The only remaining problem is to generate the lune list for each face. This can be done with the preprocessing time of hyperplanes [SC91c]. Hence follows the theorem.

Theorem 5.6 *A* β*-skeleton for* $\beta > 2$ *can be computed in* $O(n^{2-a(d)}(\log n)^{1-a(d)})$ *time.*

5.4 Conclusion

In this chapter, we have generalized the notion of the β-skeleton to higher dimensions and presented an efficient algorithm for computing a lune-based β-skeleton for $\beta > 2$ in any dimension.

The problem of finding an optimal algorithm for computing β-skeletons remains open. A recent interesting discovery is that in 2-dimension a circle-based β-skeleton is subgraph of a Minimum Weight Triangulation on n points for $\beta > 1.7317$. It would be interesting to know if this result extends to higher dimensions.

Chapter 6

Conclusions

In this thesis, we have attempted to further some of the of the algorithmic aspects of the class of geometric graphs known as β-skeletons. The highlights of our contribution, in chapterwise order, are as follows.

In the first chapter, we have have shown how to compute efficiently a β-skeleton for any $\beta \geq 0$. The complexity of our algorithm is in $O(n^{3/2} \log n)$ for $\beta \geq 1$ and in $O(n^{5/2} \log n)$ for β in in the range $[0, 1)$, for any L_p, where $1 < p < \infty$. In the L_1 and L_∞ metrics, the algorithms are in $O(n^{5/2} \log n)$ for any $\beta \geq 0$.

We have also extended our algorithms to compute generalized β-skeletons, namely, $k\beta$-skeleton and additively weighted β-skeleton. The time complexity of computing the $k\beta$-skeleton is in $O(kn^{3/2} \log n + k^2 n)$ for $\beta \geq 1$ and in $O(n^{5/2} \log n + n^2 k)$ for β in the range $[0, 1)$. The time complexity of computing the weighted β-skeleton is in $O(n^{3/2} \log n)$ for $\beta \geq 1$ and in $O(n^{5/2} \log n)$ for β in the range $[0, 1)$.

In the metric L_∞, a β-lune is simply a rectangle parallel to one of the axes. In the second chapter, we have exploited this property to present an optimal output-sensitive algorithm for computing a β-skeleton for $\beta \geq 2$. The time required is in $O(n \log n + K)$, where K is size of the output graph.

In the third chapter, we have presented an efficient algorithm for computing the β-spectrum. Our algorithms are in $O(n^{3/2} \log\ n + K)$ for $\beta_{max} \geq 1$ and in $O(n^2 \log\ n + K)$ for $\beta_{max} \geq 0$, where K is a count of the number of times that the edges of the initial (Delaunay and complete respectively) graph are witnessed for $\beta = \infty$.

We have also extended the notion of the β-spectrum to $k\beta$-spectrum and additively weighted β-spectrum. We have described fast algorithms for computing these graphs. The time complexity for computing the $k\beta$-spectrum is in $O(kn^{3/2} \log\ n+K)$ for $\beta_{max} \geq 1$ and in $O(n^2 \log\ n + K)$ time for $\beta_{max} \geq 0$, where K is a count of the number of times that the edges of the initial (k-Delaunay and complete respectively) graph are witnessed for $\beta = \infty$. The algorithm for computing the weighted β-spectrum takes $O(n^{3/2} \log\ n + K)$ time for $\beta_{max} \geq 1$ and $O(n^2 \log\ n + K)$ time for $\beta_{max} \geq 0$, where K is a count of the number of times that the edges of the initial (weighted Delaunay and complete respectively) graph on P are witnessed for $\beta = \infty$.

We have extended the notion of circle-based and lune-based β-skeletons to higher dimensions. For fixed d, we have shown that the size of the lune based β-skeleton in d dimension is linear for $\beta > 2$. We have also presented an efficient algorithm for computing a lune-based β-skeleton for $\beta > 2$.

6.1 Further Research Problems

The research reported in this thesis opens up many new questions.

A rather obvious one is to find more efficient algorithms for computing the various geometric graphs that we have discussed in this thesis, viz., β-skeleton, $k\beta$-skeleton, weighted β-skeleton etc.

In the metrics L_1 and L_∞, computing the β-skeleton, for β in [1,2), in time lower

order than $O(n^2 \log n)$ is an interesting open question. It is also worth investigating if the output-sensitive algorithm presented in this thesis for computing the β-skeleton can be generalized to higher dimensions for $\beta \geq 2$.

One really interesting avenue that we haven't explored is the application of β-skeletons to practical problems. We hope that somebody will take up this applca-tion challenge. Like, for example, using β-skeletons for shape reconstruction and identification of real world objects.

Recently, researchers have dicsovered an intriguing link between circle-based β-skeletons and Minimum Weight Triangulations [Kei94, Yan95, CX96]. For a planar point set, a circle-based β-skeleton is a subgraph of the Minimum Weight Triangulation for $\beta > 1.17682$. It would be worth investigating if this result generalises to higher dimensions.

Another line of work that is tantalizingly open is the problem of recognising a β-skeleton. Some preliminary work in this direction has been done by Bose *et al* [BLL96]. But a vast untapped field lies ahead.

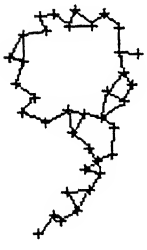
Appendix A

Some Examples of β-Skeletons

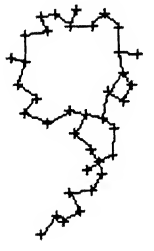
β=1.00

β=1.25

β=1.50

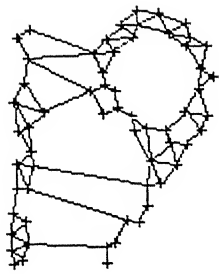


β=1.75

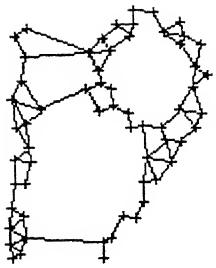


β=2.00

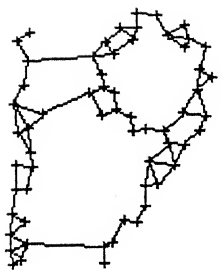
β=2.50



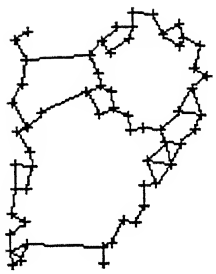
β=1.00



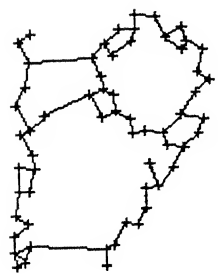
β=1.25



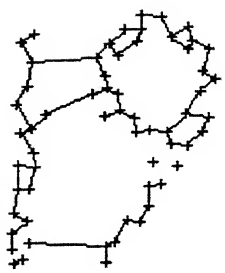
β=1.50



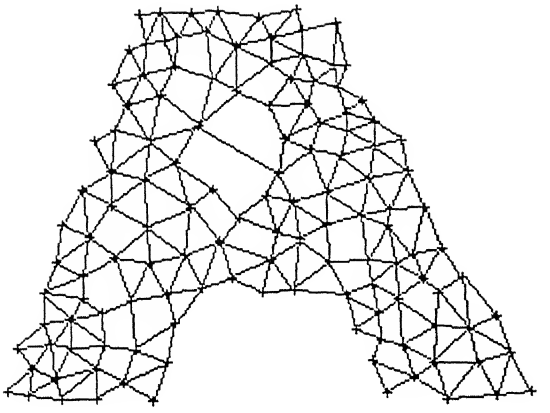
β=1.75



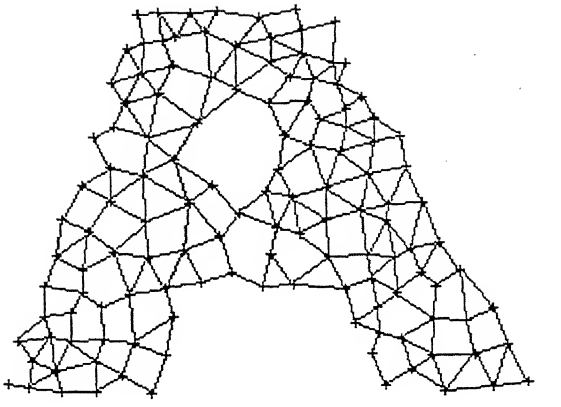
β=2.00



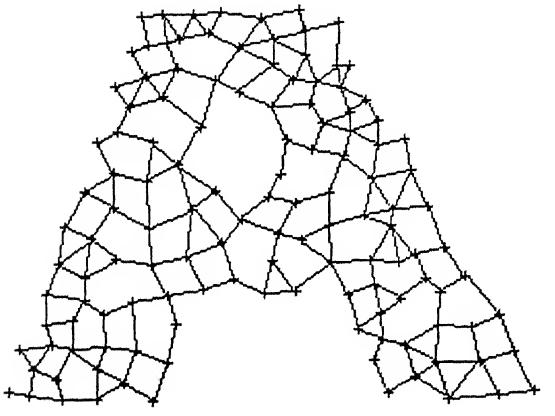
β=2.25



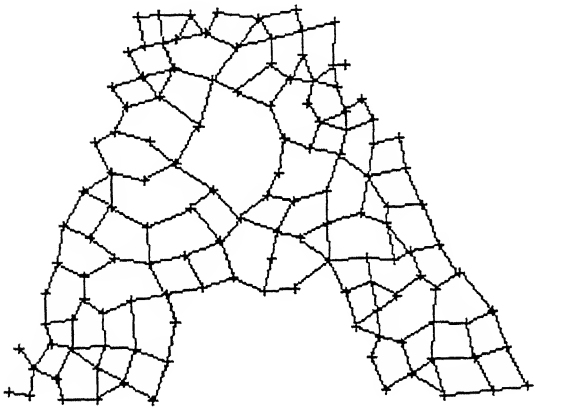
β=1.00



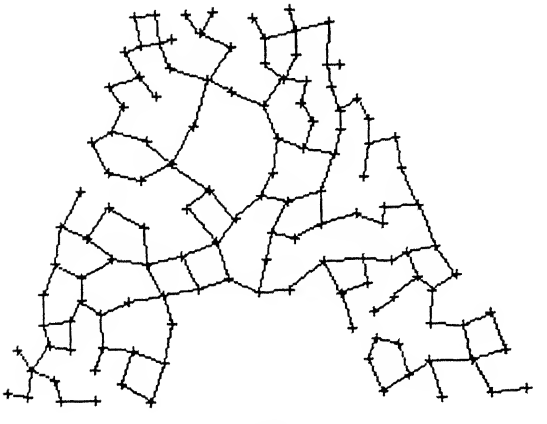
β=1.25



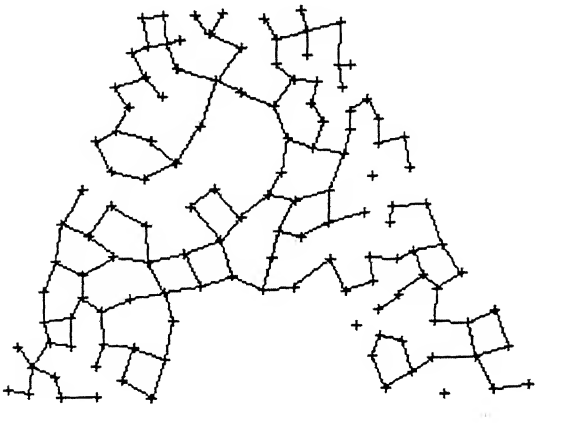
β=1.50



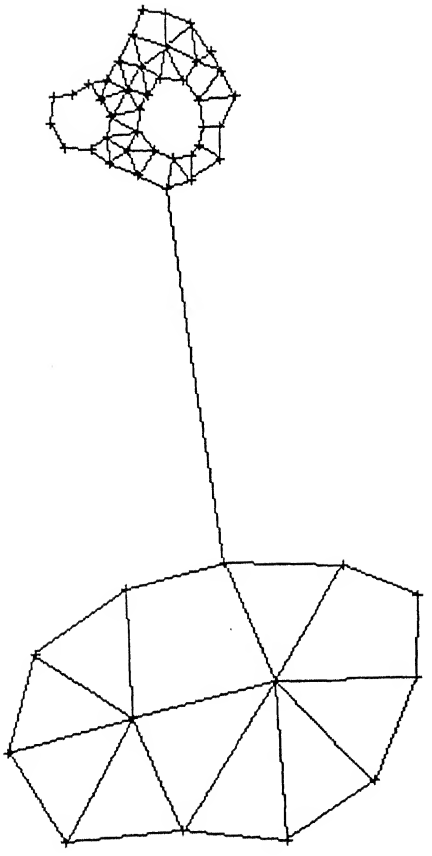
β=1.75



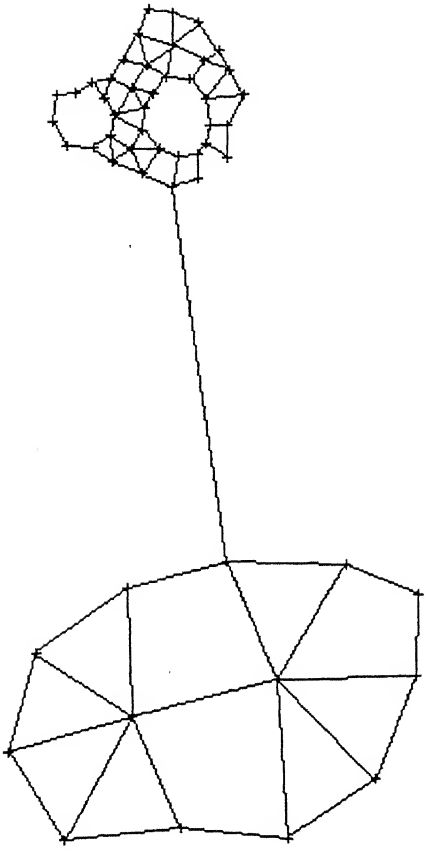
β=2.25



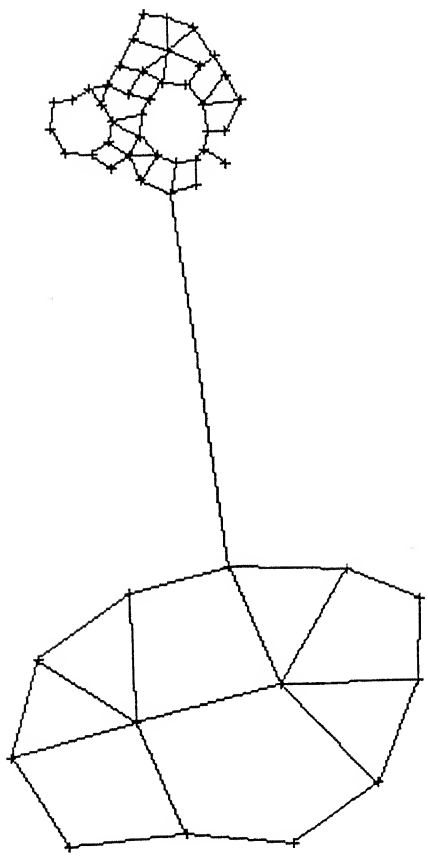
β=2.50



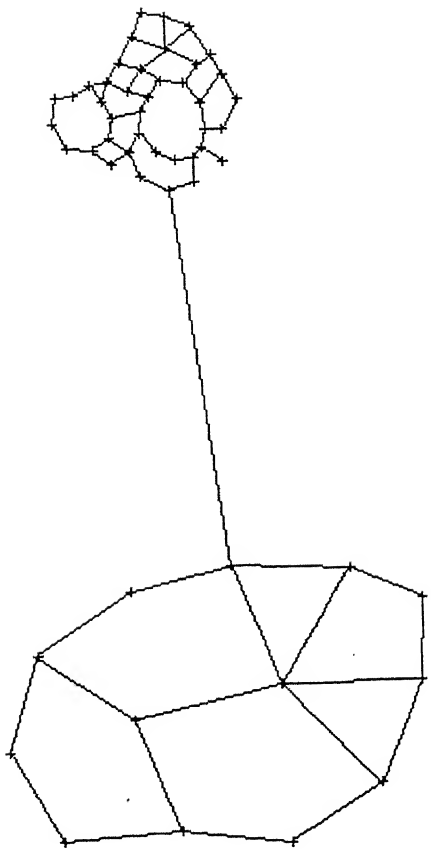
β=1.00



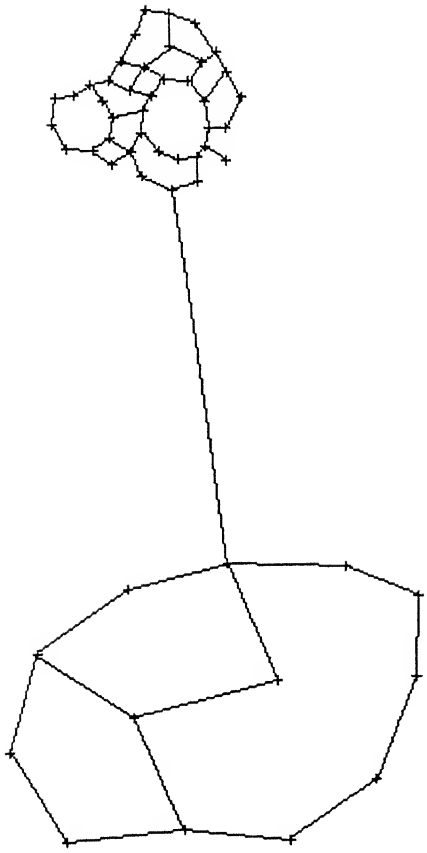
β=1.25



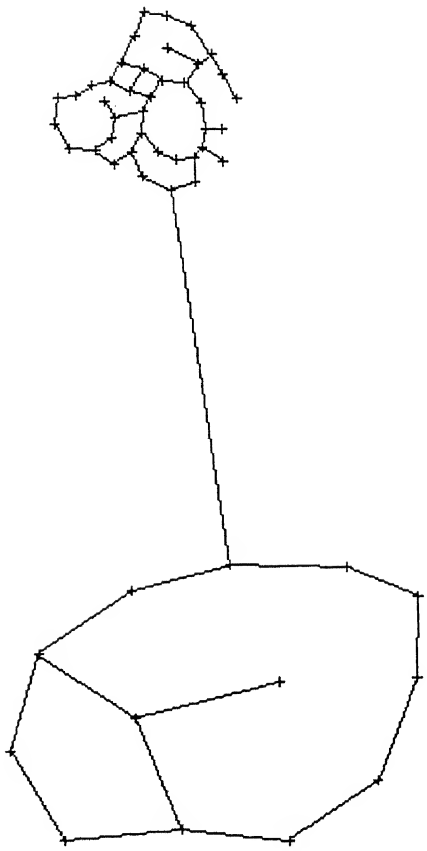
β=1.50



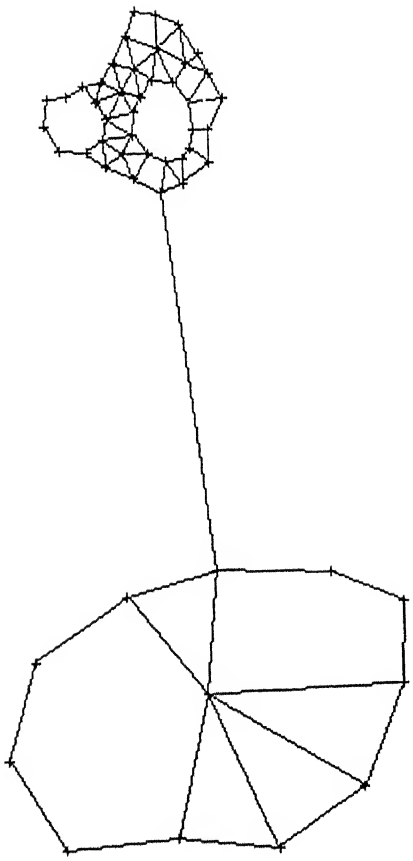
β=1.75



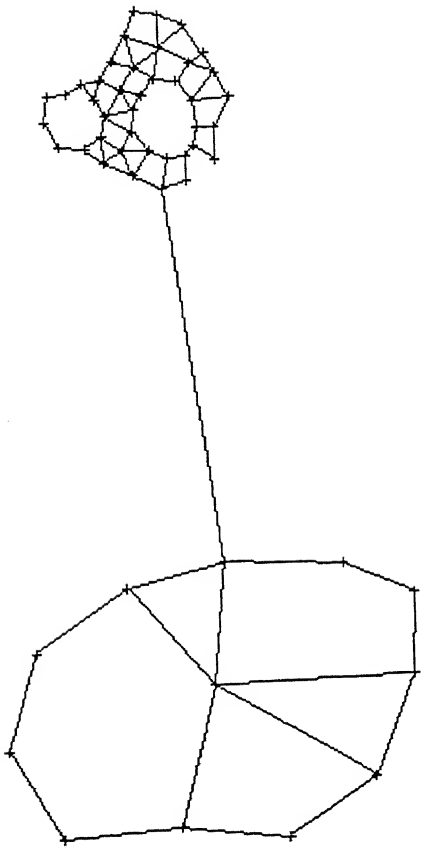
β=2.00



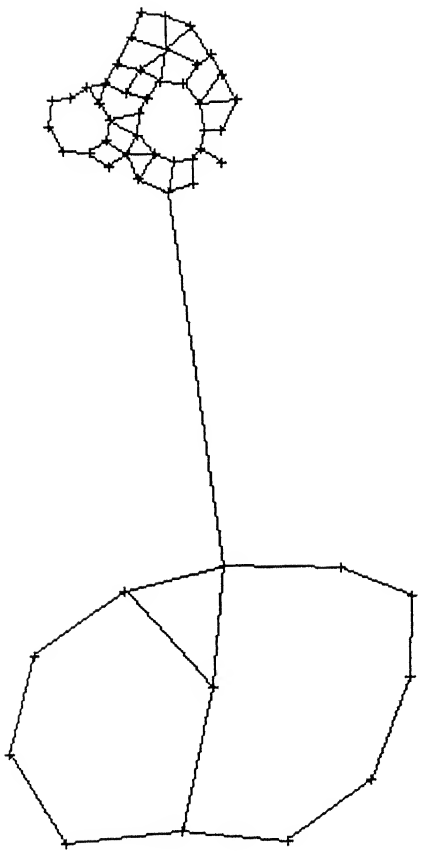
β=2.50



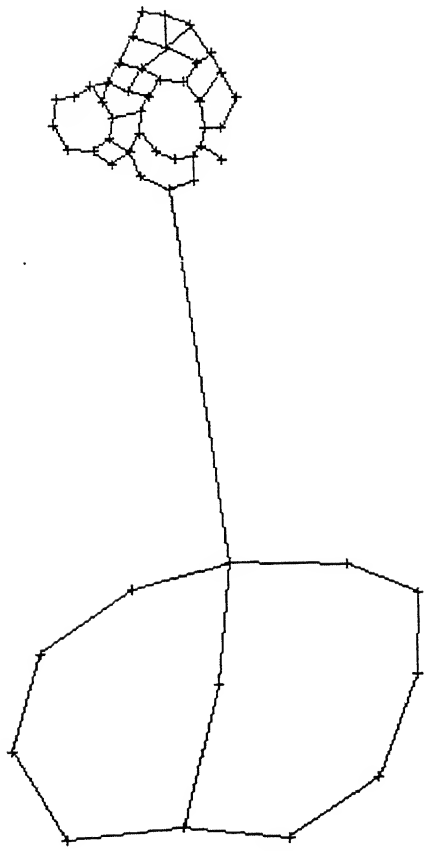
β=1.00



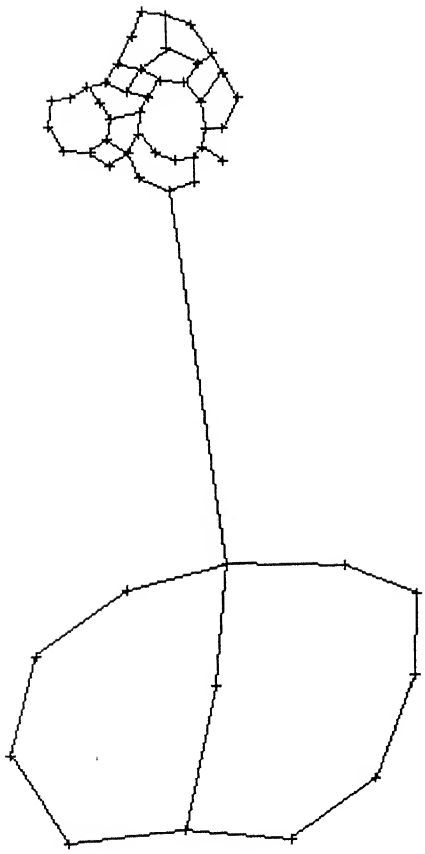
β=1.25



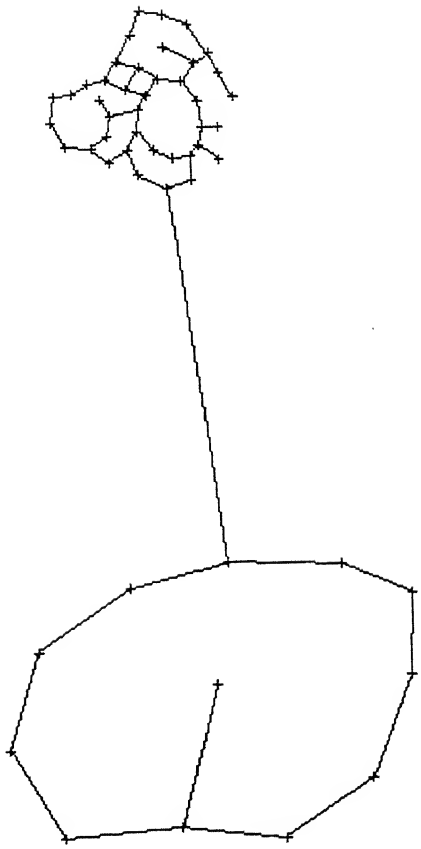
β=1.50



β=1.75



β=2.00



β=2.50

References

[ABE97] N. Amenta, M. Bern, and D. Eppstein. The crust and the β-skeleton: Combinatorial curve reconst ruction. Research Report, Xerox PARC, 1997. —http://www.geom.umn.edu/ nina/papers/crust.ps.gz—.

[AE84] F. Aurenhammer and H. Edelsbrunner. An optimal algorithm for constructing the weighted Voronoi diagram in the plane. *Pattern Recogn.*, 17:251–257, 1984.

[AESW91] P. Agarwal, H. Edelsbrunner, O. Schwarzkopf, and E. Welzl. Euclidean minimum spanning trees and bichromatic closest pairs. *Discrete Comput. Geom.*, 6:407–422, 1991.

[AJM97] Sanjay Agarwal, Gunjan Jha, and Asish Mukhopadhyay. Surface reconstruction from contour data : Improvements to the barequet-sharir heuristic. In *Proceedings of the Computer Society Convention of India*, pages 312–321, Ahmedabad, India, November 1997.

[AM92] P. K. Agarwal and J. Matousek. Relative neighborhood graphs in three dimensions. In *Proc. 3rd Annu. ACM Sympos. Discrete Algorithms*, pages 58–67, 1992.

[Att98] D. Attali. r-regular shape reconstruction from unorganized points. *Comput. Geom. Theory Appl.*, 10:239–247, 1998.

[BLL96] P. Bose, W. Lenhart, and G. Liotta. Characterizing proximity trees. *Algorithmica*, 16:83–110, 1996.

[BM79] J. L. Bentely and H. A. Maurer. A note on euclidean near neighbor searching in the plane. *Inform. Process. Lett.*, 8:133–136, 1979.

[Boi88] J.-D. Boissonnat. Shape reconstruction from planar cross-sections. *Comput. Vision Graph. Image Process.*, 44(1):1–29, October 1988.

[Bro79] K. Q. Brown. *Geometric transformations for fast geometric algorithms.* PhD thesis, Department of Computer Science, Carnegie Mellon University, Dec. 1979.

[BS94] Gill Barequet and Micha Sharir. Piecewise linear interpolation between polygonal slices. In *Proc. 10th Annu. ACM Sym. Comput. Geom.*, pages 93–101, 1994.

[BWY80] Jon Louis Bentley, Bruce W. Weide, and Andrew C. Yao. Optimal expected-time algorithms for closest point problems. *ACM Trans. on Mathematical Software*, 6(4):563–580, 1980.

[CGT95] Siu Wing Cheng, Mordecai J. Golin, and Jeffrey C. F . Tsang. Expected case analysis of β-skeletons with applications to the construction of minimum-weight triangulation. In *Proc. 7th Canad. Conf. Comput. Geom.*, pages 279–284, 1995.

[Cha86] Bernard Chazelle. Filtering search: A new approach to query-answering. *SIAM J. Computing*, 15(3):703–724, 1986.

[CTL91] M. Chang, C. Tang, and R. Lee. 20-relative neighborhood graphs are hamiltonian. *J. Graph Theory*, 15:543–557, 1991.

[CTL92a] M. Chang, C. Tang, and R. Lee. Solving the euclidean bottleneck biconnected edge subgraph problem by 2-relative neighborhood graph. *Discrete Appl. Math.*, 39:1–12, 1992.

[CTL92b] M. Chang, C. Tang, and R. Lee. Solving the euclidean bottleneck matching problem by k-relative neighborhood graph. *Algorithmica*, 8:177–194, 1992.

[CX96] Siu-Wing Cheng and Yin-Feng Xu. Approaching the largest β-skeleton within a minimum weight triangulation. In *Proc. 12th Annu. ACM Sym. Comput. Geom.*, pages 196–203, 1996.

[Del34] B. Delaunay. Sur la sphère vide. A la memoire de Georges Voronoi. *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskih i Estestvennyh Nauk*, 7:793–800, 1934.

[DH73] O. Duda and P. E. Hart. *Pattern classification and scene analysis.* Wiley-Interscience, New York, 1973.

[DL76] D. P. Dobkin and R. J. Lipton. Multidimensional searching problems. *SIAM J. Comput.*, 5:181–186, 1976.

[Dod72] C. W. Dodge. *Euclidean geometry and transformations.* Addison-Wesley, Reading, MA, 1972.

[Ede87] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*, volume 10 of *EATCS Monographs on Theoretical Computer Science.* Springer-Verlag, Heidelberg, West Germany, 1987.

[Ede92] H. Edelsbrunner. Weighted alpha shapes. Technical Report UIUCDCS-R-92-1760, Dept. Comput. Sci., Univ. Illinois, Urbana, IL, 1992.

[EGS86] H. Edelsbrunner, L. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM J. Computing*, 15:317–340, 1986.

[EKS83] H. Edelsbrunner, D. K. Kirpatrick, and R. Seidel. On the shape of a set of points in the plane. *IEEE Tran. Inform. Theory*, 29(4):551–559, 1983.

[EM94] H. Edelsbrunner and E. P. Mucke. Three-dimensional alpha shapes. *ACM Trans. Graphics*, 13(1):43–72, 1994.

[ET88] H. A. ElGindy and G. T. Toussaint. Computing the relative neighborhood decomposition of a simple polygon. In *Computational Morphology*, pages 53–70. Elsevier Science Publishers, North-Holland, 1988.

[For87] S. Fortune. A sweepline algorithm for voronoi diagrams. *Algorithmica*, 2:153–174, 1987.

[GB78] A. Getis and B. N. Boots. *Models and spatial process.* .Cambridge University Press, 1978.

[GKDM96] Vishal Goenka, Deepak Kumar, Nabanjan Das, and Asish Mukhopadhyay. Surface reconstruction from parallel planar contours. In *Proceedings of the Computer Society of India Convention*, pages 441–444, Bangalore, India, November 1996.

[GS83] L. J. Guibas and J. Stolfi. On computing all north-east nearest neighbors in the l_1 metric. *Information Processing Letters*, 17:219–223, 1983.

[HCF77] P. Haggett, A. D. Cliff, and A. Frey. *Locational Analysis in Human Geography.* Arnold, London, 1977.

[Hwa90] N. F. Hwang. Divide and conquer algorithms for rng. *BIT*, 30:196–206, 1990.

[IS85] M. Ichino and J. Sklansky. The relative neighborhood graph for mixed feature variable. *Pattern Recogn.*, 18:161–167, 1985.

[JK87] J. W. Jaromczyk and M. Kowaluk. A note on relative neighborhood graphs. In *Proc. 6th ACM Sympos. Comput. Geom.*, pages 233–241, 1987.

[JK91] J. W. Jaromczyk and M. Kowaluk. Constructing relative neighborhood graph in 3-dimensional euclidean space. *Discrete Appl. Math.*, 31:181–192, 1991.

[JKY] J. W. Jaromczyk, M. Kowaluk, and F. Yao. An optimal algorithm for constructing β-skeletons in the l_p metric. To be published in SIAM J. Computing.

[JT92] J. W. Jaromczyk and Godfried T. Toussaint. Relative neighborhood graphs and their relatives. In *Proc. IEEE*, pages 1502–1517, 1992.

[Kat88] J. Katajainen. A region approach for computing relative neighborhood graph in the l_p metric. *Computing*, 40:147–161, 1988.

[Kei94] M. Keil. Computing a subgraph of the minimum weight triangulation. *Comput. Geom. Theory Appl.*, 4:13–26, 1994.

[KN86] J. Katajainen and O. Nevalainen. Computing relative neighborhood graphs in the plane. *Pattern Recogn.*, 19:221–228, 1986.

[KN87] J. Katajainen and O. Nevalainen. An almost naive algorithm for finding relative neighborhood graphs in l_p metrics. *RAIRO Informatique Theorique et Applications*, 21:199–215, 1987.

[KNT87] J. Katajainen, O. Nevalainen, and J. Teuhola. A linear expected time algorithm for computing relative neighborhood graphs. *Inform. Process. Lett.*, 25:77–86, 1987.

[Koe90] Jan Koenderink. *Solid Shape*. MIT Press, 1990.

[KR85] D. G. Kirkpatrick and J. D. Radke. A framework for computational morphology. In G. T. Toussaint, editor, *Computational Geometry*, pages 217 248. North-Holland, 1985.

[Lee85] D. T. Lee. Relative neighborhood graphs in the l_1-metric. *Pattern Recogn.*, 18:327–332, 1985.

[Lin94] Andrzej Lingas. A linear-time construction of the relative neighborhood graph from the delaunay triangulation. *Comput. Geom. Theory Appl.*, 4:199–208, 1994.

[LM95] Andrzej Lingas and Asish Mukhopadhyay. A linear-time construction of the relative neighborhood graph within a histogram. In *Proc. 4th Workshop Algorithms Data Struct.*, volume 955 of *Lecture Notes Comput. Sci.*, pages 228–238. Springer-Verlag, 1995.

[Meg83] N. Megiddo. Linear-time algorithms for linear programming in r^3 and related problems. *SIAM J. Computing*, 12:759–776, 1983.

[Mel97] Mahmoud Melkemi. A-shapes of a finite point set. In *Proc. 13th Annu. Symp. Comput. Geom.*, pages 367–369, 1997.

[Mir93] Andy Mirzaian. Minimum weight euclidean matching and weighted relative neighborhood graphs. In *Proc. 3rd Workshop Algorithms Data Struct.*, volume 709 of *Lecture Notes Comput. Sci.*, pages 506–517, 1993.

[MS84] D. W. Matula and R. R. Sokal. Properties of gabriel graphs relevant to geographical variation research and the culturing of points in the plane. *Geographical Analysis*, 12:205–222, 1984.

[Ola89] S. Olariu. A simple linear-time algorithm for computing the rng and mst of unimodal polygon. *Inform. Process. Lett.*, 31:243–248, 1989.

[O'R82] J. O'Rourke. Computing the relative neighborhood graph in the l_1 and l_∞ metrics. *Pattern recogn.*, 15:189–192, 1982.

[Pos81] M. J. Post. A minimum spanning ellipse algorithm. In *Proc. 22nd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 115–122, 1981.

[Pos84] M. J. Post. Minimum spanning ellipsoids. In *Proc. 16th Annu. ACM Sympos. Theory Comput.*, pages 108–116, 1984.

[PS85] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer, New York, 1985.

[Rad88] J. D. Radke. On shape of a set of points. In G. T. Toussaint, editor, *Computational Morphology*, pages 105–136. North-Holland, 1988.

[SC90] T. H. Su and R. Ch. Chang. The k-gabriel graphs and their applications. In *Proc. Int. Symp., SIGAL '90*, pages 66–75, 1990.

[SC91a] T. H. Su and R. Ch. Chang. Computing the constrained relative neighborhood graphs and gabriel graphs in euclidean plane. *Pattern Recogn.*, 24:221–230, 1991.

[SC91b] T. H. Su and R. Ch. Chang. Computing the k-relative neighborhood graphs in euclidean plane. *Pattern Recogn.*, 24:231–239, 1991.

[SC91c] T. H. Su and R. Ch. Chang. On constructing relative neighborhood graphs in euclidean k-dimensional spaces. *Computing*, 46:121–130, 1991.

[Sha85] M. Sharir. Intersection and closest-pair problems for a set of planar discs. *SIAM J. Comput.*, 14:448–468, 1985.

[Sup83] K. J. Supowit. The relative neighborhood graph with an application to minimum spanning trees. *J. ACM*, 30:428–448, 1983.

[TM80] G. T. Tousaint and R. Menard. Fast algorithms for computing the planar relative neighborhood graph. In *Proc. 5th Sympos. Operations Research*, pages 425–428, 1980.

[Tou80a] G. T. Toussaint. Decomposing a simple polygon with the relative neighborhood graph. In *Proc. Allerton Conf.*, pages 20–28, October 1980.

[Tou80b] G. T. Toussaint. Pattern recognition and geometrical complexity. In *Proc. 5th Int. Conf. Pattern Recogn.*, pages 1324–1347, 1980.

[Tou80c] G. T. Toussaint. The relative neighborhood graph of a finite planar set. *Pattern Recogn.*, 12:261–268, 1980.

[Tou88] G. T. Toussaint. A graphy-theoretical primal sketch. In *Computational Morphology*, pages 229–260. Elsevier Science Publishers, North-Holland, 1988.

[TY88] J. I. Toriwaki and S. Yokoi. Voronoi and related neighbors on digitized two dimensional space with application to texture analysis. In G. T. Toussaint, editor, *Computational Morphology*, pages 207–228. North-Holland, 1988.

[Urq80] R. B. Urquhart. Algorithms for computation of relative neighborhood graph. *Electron. Lett.*, 14:556–557, 1980.

[Urq83] R. B. Urquhart. Some properties of the planar euclidean relative neighborhood graph. *Pattern Recogn. Lett.*, pages 317–322, 1983.

[Vai89] P. M. Vaidya. An $o(n \log n)$ algorithm for the all nearest neighbors problem. *Discrete Comput. Geom.*, 4:101–115, 1989.

[VBW94] A. Varshney, F. P. Brooks, Jr., and W. V. Wright. Interactive visualization of weighted three-dimensional alpha hulls. In *Proc. 10th Annu. ACM Sympos. Comput. Geom.*, pages 395–396, 1994.

[Vel92a] R. C. Veltkamp. *Closed object boundaries from scattered points.* PhD thesis, Center for Mathematics and Computer Science, Amsterdam, 1992.

[Vel92b] R. C. Veltkamp. The γ-neighborhood graph. *Comput. Geome. Theory Appl.*, 4:227–246, 1992.

[Vel94] R. C. Veltkamp. *Closed Object Boundaries from Scattered Points*, volume 885 of *Lecture Notes Comput. Sci.* Springer-Verlag, 1994.

[Vel95] R. C. Veltkamp. Boundaries through scattered points of unknown density. *Graphics Models and Image Processing*, 57(6):441–452, November 1995.

[Vin89] Luc Vincent. Graphs and mathematical morphology. *Signal Processing*, 16:365–388, 1989.

[Yan95] B. Yang. A better subgraph of the minimum weight triangulation. In *Proc. Internat. Conf. Comput. Combinatorics*, pages 452–455, 1995.

[Yao82] A. C. C. Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM J. Comput.*, 11:721–736, 1982.

List of Publication Based on This Thesis

1. S. V. Rao and Asish Mukhopadhyay. An efficient algorithm for computing β-skeleton and its generalization. In *13th European workshop on computational Geometry*, March 20-21, 1997, University of Wuerzburg, Germany.

2. S. V. Rao and Asish Mukhopadhyay. Efficient algorithm for computing the β-spectrum. In *7th National seminar on Theoretical Computer Science*, June 11-14, 1997, Chennai, India.

3. S. V. Rao and Asish Mukhopadhyay. Fast algorithms for computing β-skeleton and their relatives. In *Proc. 8th Annual international symposium on algorithms and computation*, LNCS 1350, Pages 374-383, December 17-19, 1997, Singapore.

4. Asish Mukhopadhyay and S. V. Rao. Output-Sensitive Algorithm for Computing the β-Skeleton. In *10th Canadian Conference on Computational Geometry (CCCG '98)* August 10 - 12, 1998, McGill University, Canada.

5. Asish Mukhopadhyay and S. V. Rao. Computing β-skeletons in Higher Dimensions. In *Indian Conference on Computer Vision, Graphics and Image Processing*, December 1998, Indian Institute of Technology, New Delhi, India.